

JAPAN PATENT OFFICE

This is to certify that the annexed is a true copy of the following application as filed with this Office.

Date of Application: March 28, 2003

Application Number: Patent Application
No. 2003-092611
[ST.10/C]: [JP2003-092611]

Applicant(s) : FUJITSU LIMITED

August 18, 2003

Commissioner,
JAPAN Patent Office Yasuo IMAI

Certificate No. P2003-3067306

312.360.0080
1503.68506

日 本 国 特 許 庁
JAPAN PATENT OFFICE

別紙添付の書類に記載されている事項は下記の出願書類に記載されている事項と同一であることを証明する。

This is to certify that the annexed is a true copy of the following application as filed with this Office.

出 願 年 月 日 2 0 0 3 年 3 月 2 8 日
Date of Application:

出 願 番 号 特 願 2 0 0 3 - 0 9 2 6 1 1
Application Number:
[ST. 10/C] : [J P 2 0 0 3 - 0 9 2 6 1 1]

出 願 人 富 士 通 株 式 会 社
Applicant(s):

2 0 0 3 年 8 月 1 8 日

特許庁長官
Commissioner,
Japan Patent Office

今 井 康 夫



出証番号 出証特 2 0 0 3 - 3 0 6 7 3 0 6

【書類名】 特許願

【整理番号】 0253798

【提出日】 平成15年 3月28日

【あて先】 特許庁長官殿

【国際特許分類】 G06F 17/16

【発明の名称】 共有メモリ型スカラ並列計算機用固有値問題の並列処理方法

【請求項の数】 5

【発明者】

【住所又は居所】 神奈川県川崎市中原区上小田中4丁目1番1号 富士通株式会社内

【氏名】 中西 誠

【特許出願人】

【識別番号】 000005223

【氏名又は名称】 富士通株式会社

【代理人】

【識別番号】 100074099

【住所又は居所】 東京都千代田区二番町8番地20 二番町ビル3F

【弁理士】

【氏名又は名称】 大菅 義之

【電話番号】 03-3238-0031

【選任した代理人】

【識別番号】 100067987

【住所又は居所】 神奈川県横浜市鶴見区北寺尾7-25-28-503

【弁理士】

【氏名又は名称】 久木元 彰

【電話番号】 045-573-3683

【先の出願に基づく優先権主張】**【出願番号】** 特願2002- 97835**【出願日】** 平成14年 3月29日**【手数料の表示】****【予納台帳番号】** 012542**【納付金額】** 21,000円**【提出物件の目録】****【物件名】** 明細書 1**【物件名】** 図面 1**【物件名】** 要約書 1**【包括委任状番号】** 9705047**【プルーフの要否】** 要

【書類名】 明細書

【発明の名称】 共有メモリ型スカラ並列計算機用固有値問題の並列処理方法

【特許請求の範囲】

【請求項 1】 共有メモリ型スカラ並列計算機用固有値問題の並列処理方法を実現させるプログラムであって、

演算すべき実対称行列あるいはエルミート行列をブロック化し、ブロック化した部分をメモリの作業域にコピーして、ブロック間の積を利用して 3 重対角化するステップと、

3 重対角化された行列から固有値と固有ベクトルを求めるステップと、

該 3 重対角化された行列から求められた固有ベクトルを、所定のブロック幅の行列演算の並列演算となるように変形されたハウスホルダー変換によって変換し、元の行列の固有ベクトルを求めるステップと、

を備えることを特徴とする並列処理方法を共有メモリ型スカラ並列計算機に実現させるプログラム。

【請求項 2】 前記 3 重対角化ステップにおいて、ブロック化された部分の更新を再帰的プログラムで行うことを特徴とする請求項 1 に記載のプログラム。

【請求項 3】 前記 3 重対角化ステップにおいて、ブロック化された部分は、キャッシュメモリのページを跨らないように小さなブロックに分割して格納し、各プロセッサで並列に演算を行うことによって処理されることを特徴とする請求項 1 に記載のプログラム。

【請求項 4】 前記固有ベクトルを求めるステップにおいて、前記ハウスホルダー変換のための行列は、各プロセッサで処理できるような小さな係数行列となる上三角行列を各プロセッサで同時に生成されることを特徴とする請求項 1 に記載のプログラム。

【請求項 5】 前記固有ベクトルを求めるステップにおいて、前記固有ベクトルは、格納されている 2 次元配列の 2 次元目をプロセッサの数に対応して均等に分割し、おのおのの分割された領域を各プロセッサに割り当てることによって、演算されることを特徴とする請求項 1 に記載のプログラム。

【発明の詳細な説明】**【0001】****【発明の属する技術分野】**

本発明は、共有メモリ型スカラ並列計算機における行列の演算に関する。

【0002】**【従来の技術】**

ベクトル並列計算機用実対称行列（行列要素が実数で、転置しても行列が不変に保たれる行列）及びエルミート行列（行列要素が複素数で、共役・転置をとっても不変に保たれる行列）の固有値問題（ A を行列、 λ を定数、 I を単位行列とした場合、 $\det | A - \lambda I | = 0$ となる λ とその固有ベクトルを求めること）は、3重対角化（対角成分とその両隣の成分のみを持つ行列にすること）をまず行う。この3重対角行列の固有値問題をマルチセクション法によって解き、固有値を求めてから逆反復法で固有ベクトルを求める。その後、固有ベクトルにハウスホルダー変換を行い、元の固有値問題の固有ベクトルを求めている。

【0003】**【発明が解決しようとする課題】**

ベクトル計算機においては、メモリアクセス系の動作速度が速いことを前提として、固有値問題の演算を行っていたが、共有メモリ型スカラ計算機の場合、演算する行列が大きくなるほどに、共有メモリにアクセスする回数が多くなり、アクセス速度の遅い共有メモリへのアクセスによって、計算機の性能が大きく損なわれてしまうという問題がある。そこで、共有メモリ型スカラ計算機の各プロセッサに設けられる、アクセス速度の速いキャッシュメモリを有効に使い、行列計算を行う必要がある。つまり、行列の各行あるいは列毎に演算していると、共有メモリへのアクセスが多くなってしまうので、行列をブロック化して、各プロセッサにキャッシュメモリに格納されたデータを最大限処理した後、共有メモリにアクセスするようにして、共有メモリへのアクセス数を減少する、各プロセッサに局所化したアルゴリズムが必要となる。

【0004】

すなわち、共有メモリ型の並列計算機では、ベクトル計算機のような強力なメ

モリアクセス能力がないため、メモリアクセスに対して演算量を増やすアルゴリズム上の工夫が必要になる。

【0005】

本発明の課題は、共有メモリ型スカラ計算機において、高速に固有値問題の演算が行える演算の並列処理方法を提供することである。

【0006】

【課題を解決するための手段】

本発明の並列処理方法は、共有メモリ型スカラ並列計算機用固有値問題の並列処理方法を実現させるプログラムであって、演算すべき実対称行列あるいはエルミート行列をブロック化し、ブロック化した部分をメモリの作業域にコピーして、ブロック間の積を利用して3重対角化するステップと、3重対角化された行列から固有値と固有ベクトルを求めるステップと、該3重対角化された行列から求められた固有ベクトルを、所定のブロック幅の行列演算の並列演算となるように変形されたハウスホルダー変換によって変換し、元の行列の固有ベクトルを求めるステップとを備えることを特徴とする。

【0007】

本発明によれば、固有値問題を共有メモリ型スカラ計算機の各プロセッサにできるだけ局所化した演算で解くことができるので、各プロセッサが、共有メモリに頻繁にアクセスするために生じる遅延時間が最小限に抑えられ、並列計算の威力を有効に発揮した演算が行える。

【0008】

【発明の実施の形態】

本発明の実施形態においては、上記固有値問題の演算における3重対角化部分のアルゴリズムでは、ブロック化されたアルゴリズムを採用し、ブロック部分を求める部分のアルゴリズムを再帰的に行うようにし、更新部分での演算密度を高める。また、メモリのアクセスがキャッシュメモリのページ間をとびとびにアクセスすることを避けるように行列ベクトル積部分を対称性を利用して連続アクセスが使える方式にする。キャッシュメモリのページを跨ってデータを読み出す場合、一度に読み込めず、キャッシュメモリに2度アクセスする必要があるの

で、計算機のパフォーマンスを落とすことにある。従って、上記のように、ページをデータがまたがないようにする。

【0009】

また、3重対角化された行列の固有ベクトルにハウスホルダー変換を行い、元の行列の固有ベクトルを求める部分は、ハウスホルダー変換を80回程度ごとに束ねて、3つの行列要素からなる演算を行うことで演算密度を高める。

【0010】

なお、本発明の実施形態においては、3重対角化された行列から固有値を求める方法及び3重対角化された行列の固有ベクトルを求める方法は、従来の方法によるとする。

【0011】

図1は、本発明の実施形態が前提とする共有メモリ型スカラ計算機のハードウェア構成を示した図である。

プロセッサ10-1～10-nは、1次キャッシュメモリを持っており、この1次キャッシュメモリはプロセッサの中に組み込まれていることもある。また、各プロセッサ10-1～10-nには、2次キャッシュメモリ13-1～13-nが設けられ、2次キャッシュメモリ13-1～13-nが相互結合網12に結合されている。また、相互結合網12には、共有メモリであるメモリモジュール11-1～11-nが設けられ、プロセッサ10-1～10-nは、演算に必要なデータをここから読み出し、相互結合網12を介して、2次キャッシュメモリ13-1～13-nあるいは、1次キャッシュメモリに記憶させて、演算を行う。

【0012】

この場合、メモリモジュール11-1～11-nから2次キャッシュメモリ13-1～13-nあるいは1次キャッシュメモリにデータを読み込んだり、2次キャッシュメモリ13-1～13-n、あるいは、1次キャッシュメモリから演算後のデータをメモリモジュール11-1～11-nに書き込むのはプロセッサ10-1～10-nの演算速度に比べて非常に遅い。従って、このような書き込み、読み出しが頻繁に発生すると計算機全体の性能を劣化させてしまう。

【0013】

従って、計算機全体の性能を高く維持するためには、メモリモジュール $11-1 \sim 11-n$ へのアクセスをできるだけ減らし、2次キャッシュメモリ $13-1 \sim 13-n$ 、1次キャッシュメモリ、及びプロセッサ $10-1 \sim 10-n$ からなるローカルな系においてできるだけ多くの演算をしてしまうようなアルゴリズムが必要となる。

<固有値、固有ベクトルを求める方式>

1. 3重対角化部分

1) 3重対角化

a) ブロック化された3重対角化の数学的アルゴリズム

行列をブロック幅ごとに3重対角化する。行列をブロックに分割してブロック単位で下記のアロリズムで3重対角化を行う。

【0014】

図2～図11は、本発明の実施形態のアロリズムを説明する図である。

図2は、 m 番目のブロックを処理しているときの図である。ブロックとは図2で点線で示した列及び行方向の矩形である。

【0015】

最後のブロックは、そのブロック幅-2をブロック幅とする、右下に 2×2 行列を残した左詰めめのブロックに関してアロリズムを適用して全ての処理を終わる。

【0016】

do $i=1, \text{blks}$

step1: A_n の $n+1$ 行ベクトルよりハウスホルダーベクトル u を作る。

step2: $v_i = A_{n+i}u$, $w_i = v_i - u(u^t v) / 2$ を計算する。

step3: $U_i = (U_{i-1}, u_i)$, $W_i = (W_{i-1}, w_i)$ と更新する。(ここで、 (U_{i-1}, u_i) は、行列 U_{i-1} という1列を加えて行列 U_i とすることにより、行列を1列ずつ拡大する。)

step4: if($i < \text{blks}$) then

A_n の $n+i+1$ 列目を更新する。

【 0 0 1 7 】

$$A_n(*, n+i+1) = A_n(*, n+i+1) - U_i W_i (n+i+1)^t - W_i U_i (n+i+1, *)^t$$

endif

enddo

$$\text{step5: } A_{n+\text{blks}} = A_n - U_{\text{blks}} W_{\text{blks}}^t - W_{\text{blks}} U_{\text{blks}}^t$$

・ブロック化されたハウスホルダー変換による 3 重対角化

・ ・ ハウスホルダー変換の説明

$$v = (v_1, v_2, \dots, v_n)$$

$$|v|^2 = v^* v = h^2$$

$U_v = (h, 0, \dots, 0)^t$ とすると、 $U_v = v - (v_1 - h, v_2, \dots, v_n)$ の関係がある。

$$U = (1 - uu^t / |u|^2) = (1 - \alpha uu^t), \text{ ここで } u = (v_1 - h, v_2, \dots, v_n)$$

以下、 α を省略して計算する。

$$\begin{aligned} A_{n+1} &= U^t A_n U = (1 - uu^t) A (1 - uu^t) \\ &= A_n - uu^t A_n - A_n uu^t + uu^t A_n uu^t \\ &= A_n - uw^t - uu^t u^t v / 2 - wu^t u^t v / 2 + uu^t u^t v \\ &= A_n - uw^t - wu^t \quad \dots \dots \dots (*) \end{aligned}$$

ここで、 $w = v - u(u^t v) / 2$, $v = A_n u$

これを繰り返し利用すれば、

$$A_{n+k} = A_n - U_k W_k^t - W_k U_k^t \quad \dots \dots \dots (**)$$

k ステップ目の計算は (*) 及び (**) から V_n を次のように計算できる。

$$v_k = A_n u_k - U_{k-1} W_{k-1}^t u_k - W_{k-1} U_{k-1}^t u_k \quad \dots \dots \dots (***)$$

$$w_k = v_k - u_k u_k^t v_k / 2$$

$$U_k = (U_{k-1}, u_k), \quad W_k = (W_{k-1}, w_k)$$

$$A_{n+k} = A_n - U_k W_k^t - W_k U_k^t$$

b) ハウスホルダー変換を構成する情報の格納

固有ベクトルを計算するときに 3 重対角化で使ったハウスホルダー変換が必要になる。このため U_n および α をハウスホルダー変換を構成するベクトルの位置に

格納する。 α は対応する対角要素の位置に格納する。

c) U_i を効率的に求める方法

ブロック部分の3重対角化を行うためにハウスホルダー変換のための次のベクトルを更新する必要がある。これらの計算をできるだけローカルに行うために、ブロック幅部分を作業域にコピーして3重対角化を行い、元の領域に格納する。次の列ベクトルの更新をそのたび行わず、行列積の形にして演算密度を上げて計算を行う。このため、ブロック部分の3重対角化を再帰的なプログラミングで行う。

【0 0 1 8】

recursive subroutine trid (幅、ブロック領域ポインタ)

if(幅<10) then

c 当該幅のブロック部分の3重対角化を行う。

【0 0 1 9】

ハウスホルダー変換で必要なベクトル u 、及び行列ベクトル積から v_i および w_i を作成する。

U 、 W に u_i 、 w_i を結合する。

【0 0 2 0】

else

c ブロック幅を半分に分割

c 前半ブロックに関して3重対角化を行う。

【0 0 2 1】

call trid (前半の幅、前半の領域)

c ブロックを分割して、分割線から後半部分を更新

$B=B-UW^t-WU^t$ で更新する。

c その後、後半部分を3重対角化する。

【0 0 2 2】

call trid (後半の幅、後半の領域)

return

end

図3に示されるように、作業領域Uにブロック部分をコピーして、ブロック部分を再帰的プログラムで3重対角化する。再帰的プログラムのため、図3の前半部分は前半の更新処理のために当該再帰的プログラムが呼び出されたときに3重対角化される。後半部分は、前半部分によって更新され、その後、3重対角化される。

【0023】

図4に示されるように、再帰的プログラムが深さ2まで呼ばれると、最初の前半処理で図4(a)の斜線部分をBとして更新し、次に、図4(c)の斜線部分、最後に図4(e)が更新される。更新時の並列化は、更新部分のブロック行列を行ベクトル方向に均等に分割して、各部分の更新を複数のプロセッサで並列に行う。

図4(a)の演算後、図4(b)の演算が行われ、図4(c)の演算後、図4(d)の演算が行われ、図4(e)の演算後、図4(f)の演算が行われる。

【0024】

図5に示されるように、Uの斜線部分を更新するとき、uの横線及びWの縦線部分を参照する。このことで、演算密度を上げることができる。すなわち、

$$A_{n+k} = A_n - U_k W_k^t - W_k U_k^t \quad \dots\dots (**)$$

を利用して、 V_n を次のように計算できる。このときU、Wの参照パターンは

$$v_k = A_n u_k - U_{k-1} W_{k-1}^t u_k - W_{k-1} U_{k-1}^t u_k \quad \dots\dots (***)$$

図5(a)と(b)、(c)と(d)、(e)と(f)のUの更新の後、更新した部分の3重対角化のために v_k を計算するときには、U、Wを参照して、行列ベクトル積で計算する。参照だけであり、Uの更新と参照が共通する部分もあり、効率的に参照できる。 A_n を毎回更新せずに、このようにU、Wを使って必要な部分を更新し、(**)を使って、全体の更新の演算密度を上げて性能向上を行う。(***)は余分な演算ではあるが、ブロック幅を小さくしておく限り無視できる。

【0025】

例えば、4個のCPUで並列処理を行う場合、(***)の $W_{k-1}^t u_k$ 、 $U_{k-1}^t u_k$ の計算では、縦線での分割で並列計算を行い、その結果との積に関しては、破線の

の方向に分割して並列計算する。

$v_i = A_n u_i$ の計算の並列化

図 6 に示されるように、 A_n の対称性、つまり、 $A_n = A_n^t$ を利用して各プロセッサで 2 次元目を分割して、 v_i の部分を $A_n(*, ns:ne)^t u_i$ を各プロセッサで計算する。

2) 共有メモリ型スカラ並列計算機での並列化

a) U、W 用の格納領域は、共有メモリに確保する。また、3 重対角化を行うブロック領域は別に確保した作業領域にコピーして、その領域で 3 重対角化を行う。

【0026】

上記の再帰的プログラムの中で、並列化する部分は以下の部分

(1) ハウスホルダー変換を構築するために必要な u_i を求めるために step4 の $A_n(*, n+i+1) = A_n(*, n+i+1) - U_i W_i(n+i+1, *)^t - W_i U_i(n+i+1, *)^t$ を使って、必要なベクトルを求める部分

(2) step2 で v_i を計算する部分。この部分は $(**)$ の以下の式に u_i を作用させて計算する。

$$A_{n+k} = A_n - U_k W_k^t - W_k U_k^t$$

この計算では A_n と u_i との積及び $U_k W_k^t - W_k U_k^t$ と u_i との積を並列化する。

【0027】

ブロック部分は、作業域にコピーされており A_n の必要部分は更新されないようにする。また、 A_n の対称性を利用して列ベクトル方向に長い行列に分割して並列計算する。

(3) 再帰的プログラムで、ブロック領域の更新は、

$$A_{n+k} = A_n - U_k W_k^t - W_k U_k^t$$

を利用して更新する。そのことで (1) の計算を減らす。

3) step5 の更新部分

更新は対称性を利用して対角成分より下半分を計算する。並列化は負荷がバランスするように CPU 数を #CPU としたとき、更新すべき部分行列の格納されているサブアレイの 2 次元目を $2 \times \#CPU$ 個に均等に分割して、1 から $2 \times \#$

CPUと番号を振る。1から#CPUのCPUに関してi番目のプロセッサでi番目の部分と $2 \times \#CPU + 1 - i$ 番目の部分の更新を並列に行う。

【0028】

このあと上半分にコピーを行う。これも同様に分割して負荷をバランスさせる。このとき、対角ブロック部分以外は、キャッシュメモリのページをまたがらないようにある程度小さなブロックに分割してコピーを行う。

$A_{n+k} = A_n - U_k W_k^t - W_k U_k^t$ で更新する部分は下三角行列部分で、図7のように列ブロックにCPU数 $\times 2$ に分割して、両端からペアを作り、これを各CPUで更新する。図7は、4個のCPUが設けられている場合を記述している。

【0029】

下三角部分が更新できたら、上三角部分に1～8を同じペアで転置してu1からu8にコピーする。

このとき、小さな正方ブロックに内部を分割してキャッシュを利用して転置する。更新と同様にCPUで並列処理する。

キャッシュ上での転置で性能向上する部分の説明

図8に示されるように、1から番号順に転置変換していく。各正方形領域に関して、1の場合は下三角部分をメモリの連続領域にコピーして、それを行方向にアクセスして（転置して）1の上三角部分に格納する。2～8は正方形にコピーして、行方向に参照して列方向（上下方向）に格納領域に格納する。

2 固有ベクトルの計算

a) 基本的なアルゴリズム

ベクトル u_n を格納して、後で $(1 - 2 * uu^t / (u^t u))$ を構成して、ベクトルにかけ

【0030】

3重対角化を行うと元の固有値問題は以下ようになる。

$$Q_{n-2} \dots Q_2 Q_1 A Q_1^t Q_2^t \dots Q_{n-2}^t Q_{n-2} \dots Q_2 Q_1 x = \lambda Q_{n-2} \dots Q_2 Q_1 x$$

3重対角化した固有値問題を解いて求めた固有ベクトル y から $x = Q_1^t Q_2^t \dots Q_{n-3}^t Q_{n-2}^t y$ を計算することで変換することができる。

b) 本発明の実施形態のブロックアルゴリズム及び固有ベクトルの変換の並列計算

多数のまたは全ての固有ベクトルを求める場合は各CPUに均等に3重対角行列の固有ベクトルを対応付けて各CPUで並列に上記の変換を施す。このとき変換行列を80本程度づつまとめて変換を行う。

【0031】

各々の変換行列 Q_i^t は $1 + \alpha_i u_i u_i^t$ と書ける。これらの積を展開すると以下の形になる。

$b_{i,j}$ は最左端及び最右端の $u_i u_j$ を除いたスカラー係数をまとめたものである。

【0032】

【数1】

$$1 + \sum_{i=n}^{n+k-1} u_i \left(\sum_{j=1}^{n+k-1} b_{ij} u_j^t \right)$$

【0033】

の形になる。 $b_{i,j}$ は上三角行列になる。これを求めて $1 + U B U^t$ の形になる。このような形に変形することで、計算密度を向上させることで高速化を図る。図9は、行列Bの模式図である。

【0034】

上記の方法は3ステップであるが、メモリアクセスの関係から行列はU及びBの2つとなり、Bは比較的小さな領域にすることができると効率が良い。

$m-1$ 番目まで $b_{i,j}$ ができた後に、 $(1 + \alpha_m U_m U_m^t)$ をかけると、

【0035】

【数2】

$$1 + \sum_i u_i \left(\sum_j b_{ij} u_j^t \right) + \alpha_m U_m U_m^t + U_m \sum_i \alpha_m u_m^t u_i \left(\sum_j b_{ij} u_j^t \right)$$

【0036】

最後の項の和をi、jを入れ替えて

【0037】

【数 3】

$$U_m(\sum(\sum \alpha_m u_m^t u_i b_{ij}) u_j^t)$$

【0038】

と変更することができる。

最内の括弧の中が $b_{m,j}$ ($j=m+1, \dots, n+k$) と考えられる。 $b_{m,m}$ は α_m である。

ブロック幅の正方な作業配列 $W2$ を用意して、はじめに $\alpha_i U_i U_j^t$ を $W2(i, j)$ の上三角部分に格納する。対角要素には α_i を格納する。

【0039】

次に、以上の議論は最右下の 2×2 上三角行列から順番に 1 行ずつ上に付け加える要領で計算することができる。

行の最も右の要素から一つずつ計算していけば、 B は上三角行列であるため更新された部分の参照が無いために同じ領域で計算ができる。このように、極めて小さな領域のみを使って 3 つの行列積の真ん中に位置する係数行列を計算することができる。

【0040】

図 10 は、上記固有値を求める方法を模式的に示した図である。

ブロック幅を nbs とする。

まず、 B の上半分に内積 $\alpha_j u_i \cdot u_j$ を計算して格納する。

【0041】

対角要素には α_i を格納する。

次に、以下のように計算する。

```
do i1=nbs-2,1,-1
do i2=nbs,i1+1,-1
sum=w2(i1,i2)
do i3=i2-1,i1+1,-1
sum=sum+w2(i1,i3)*w2(i3,i2)
enddo
```

```

w2(i1, i2)=sum
enddo
enddo
do i2=nbs, 1, -1
do i1=i2-1, 1, -1
w2(i1, i2)=w2(i1, i2)*w2(i2, i2)
enddo
enddo

```

図 1 1 は、上記で求められた固有ベクトルを元の行列の固有ベクトルに変換する様子を模式的に示した図である。

【 0 0 4 2 】

配列 A に格納されているハウスホルダーベクトルで固有ベクトルを変換する。ブロック化されていて、図 1 1 の斜線の部分を E V の斜線部分にかけて結果を W に格納する。また、A ブロック行列から W 2 を構成する。W 2 と W をかけたものに A のブロック部分をかけたものから E V の斜線部分を更新する。

3 エルミート行列の固有値・固有ベクトル

エルミート行列の固有値・固有ベクトルを求めるアルゴリズムは実対称行列の 3 重対角化での転置を転置＋複素共役に置き換える（ $t \rightarrow H$ ）。また、ハウスホルダーベクトルの作り方は最初の要素のスカラ倍になるようにベクトルの大きさを加える。

【 0 0 4 3 】

できた 3 重対角行列はエルミート行列であり、これを絶対値 1 の対角行列でスケールリングする。

対角行列は以下のように作る。

$$d_i = 1.0, \quad d_{i+1} = h_{i+1, i} / |h_{i+1, i}| * d_i$$

図 1 2 ～図 1 8 は、本発明の実施形態に従ったルーチンの疑似コードである。

【0044】

図12は、実対称行列を3重対角化するサブルーチンである。

配列aに実対称行列の下三角部分を格納する。daig、sdiagに3重対角行列、副対角部分を格納する。出力としてaの下三角部分に変換に必要な情報を格納する。

【0045】

Uは3重化を行うブロックを格納する。VはWを格納する領域である。

nbは、ブロックの数であり、nbaseはブロックの始まり位置を示す。

サブルーチンcopyを実行した後、3重対角化をするブロックをu(nbase+1:n, 1:iblk)に格納し、blktridルーチンを呼び出してLU分解を行う。そして、処理の終わったu(nbase+1:n, 1:iblk)を元の行列aに書き戻す。これらの下の処理は、最後の残りのブロックの3重対角化をサブルーチンblktridを用いて行うものである。

【0046】

図13は、3重対角化を行うサブルーチンの疑似コードである。

このサブルーチンは、ブロック行列を3重対角化するルーチンで、再帰的に呼び出される。nbaseはブロックの位置を示すオフセット、istartは再帰呼び出しで対象となる縮小されたサブブロックのブロック内でのオフセットで最初1で再帰的に呼び出されるとき対象サブブロックの位置を示す。nwidthはサブブロックの大きさである。

【0047】

nwidthが10より小さい場合には、サブルーチンbtunitを呼び出し、そうでない場合には、istartをistart2に格納し、nwidthの2分の1をnwidth2に格納し、サブルーチンblktridによって3重対角化し、その後、バリア同期を取る。

【0048】

更に、istartとnwidth/2を加えたものをistart3に格納し、nwidth-nwidth/2をnwidth3に格納し、ブロックの始まりと終わりをそれぞれ示すis2、is3、ie2、ie3、is、ieに値

を設定し、`len`、`iptr`を設定し、同図の式に従って計算した後、`u(is : ie、is3 : ie3)`に格納して、バリア同期を取り、3重対角化サブルーチン`blktrid`を呼び出して処理をしてサブルーチン処理を終わる。

【0049】

図14は、3重対角化サブルーチンの内部ルーチンの疑似コードである。

3重対角化の内部サブルーチン`btunit`においては、必要な情報を格納した後、ブロックの始まり`iptr2`と、幅`len`、始まり位置`is`、終わり位置`ie`を決定し、バリア同期を取る。そして、 $u(is : ie, i)^t * u(is : ie, i)$ を`tmp`に格納し、バリア同期を取る。その後、各値を計算し、それぞれの配列に格納する。ここで`sum`は和、`sqr`は平方根をとることを意味する。そして、最後にバリア同期を取る。

【0050】

次に、 $v(is : ie, i)$ を求めて、バリア同期を取り、`len2`、`isx`、`ie x`、`u`、`v`を更新して、バリア同期を取る。更に、 $v(is : ie, i)$ を更新し、バリア同期を取る。更に、 $v(is : ie, i)^t * u(is : ie, i)$ を計算し、`tmp`に格納し、バリア同期を取る。

【0051】

そして、`beta`に値を設定し、バリア同期を取り、`v`を`beta`を使った演算で更新し、バリア同期を取る。

そして、 $i < i_{blk}$ かつ $p_{tr2} < n - 2$ である時には、 $u(is : ie, i + 1)$ を更新し、他の場合には、別の式で $u(is : ie, i ; 1 : i + 2)$ を更新し、処理を終了する。なお、このサブルーチンの実行後、スレッドの割当てを解除する。

【0052】

図15は、`u`、`v`から行列の下半分を更新するルーチン及び、対角行列部分の更新をするルーチン及びコピーのルーチンの疑似コードである。

ここで、`nbase`はブロックの位置を示すオフセットであり、`nwidth`は、ブロック幅である。

【0053】

このサブルーチン `update` では、配列 `a`、`u`、`v` を確保した後、バリア同期を取る。そして、`blk`、`nbase2`、`len`、`isl`、`iel`、`nbase3`、`isr`、`ier` を設定した後、`a(iel:n, isl:iel)`、及び、`a(ier+1:n, isr:ier)` をそれぞれ更新し、サブルーチン `trupdate` を2回呼び出し、バリア同期を取って処理を返す。

【0054】

サブルーチン `trupdate` では、対角ブロック更新用ブロック幅を `blk2` に格納し、`a`、`u`、`v` の配列を確保する。

`is2`、`ie2` を設定し、`A(is2:ie, is2:ie2)` を更新し、処理を返す。

【0055】

サブルーチン `copy` では、`len`、`isl`、`lenl`、`nbase`、`isr`、`lenr` を設定して、`bandcp` を2回実行し、処理を返す。

図16は、更新した下三角部分を上三角部分にコピーするルーチンの疑似コードである。

【0056】

サブルーチン `bandcp` では、`nb`、`w`、`nn`、`loopx` を設定し、`do` ループの中で、`TRL(w(1:nnx, 1:nnx))` に `TRL(a(is2:is2+nnx-1, is2:is2+nnx))` を格納し、`TRU(a(is2:is2+nnx-1, is:is+nnx))` に `TRL(w(i:nnx, 1:nnx))t` を格納する。ここで、`TRL` は下三角部分、`TRU` は上三角部分を示す。

そして、`w(1:nnx, 1:nnx)` と `a(is2:is2+nnx, is3:is3+nnx-1)` を更新し、次に、`w(1:ny, 1:nx)` と `a(is2:is2+nnx, is3:n)` を更新する。

【0057】

そして、`do` ループを抜けると、処理を返す。

図17は、3重対角行列の固有ベクトルを元の行列の固有ベクトルに変換するルーチンの疑似コードである。

【0058】

ここで、3重対角行列の固有ベクトルは、 $ev(1:n, 1:n_{ev})$ に格納されている。また、 a は、3重対角化の出力であり、変換に必要な情報を下三角部分に格納している。

【0059】

サブルーチン `conv ev` は、配列として a と ev を取る。

スレッドを構成し、処理を並列処理で行う。

スレッド間でバリア同期をとり、 len 、 is 、 ie 、 $n_{ev}thrd$ を設定し、`conv ev thrd` ルーチンを呼び出し、リターン後バリア同期を取って処理を終了する。

【0060】

図18は、固有ベクトルを変換するルーチンの疑似コードである。

サブルーチン `conv ev thrd` では、ブロック幅を blk に格納し、配列として a 、 ev 、 w 、 $w2$ を取る。

【0061】

まず、 $width$ が0より小さければ、処理をせず元のルーチンに戻る。 $numblk$ と n_{fbs} を設定し、 $alpha$ に3重対角化時に対角要素に格納された値の符号を逆にしたもの ($-a(i, i)$) を入れる。また、 $x(1:iwidth)$ に $ev(i+1:n, 1:iwidth)^t * a(i+1:n, i)$ を入力し、これと、 $alpha$ と a を用いて ev を更新する。更に、次の `do` 文で、 is 、 ie を設定し、 $w(1:blk, iwidth)$ を $a(is+1:n, is:ie)^t * ev(is+1:n, 1:iwidth)$ で上書きし、同じく $w(1:blk, 1:iwidth)$ を $TRL(a(ie+1:is, is:ie))^t * ev(ie+1:is, 1:iwidth)$ で更新する。ここで、 TRL は、下三角行列部分である。

【0062】

また、 $w2$ の対角要素ベクトル $DIAG(w2)$ に、 $a(is:ie, is:ie)$ の対角要素ベクトルを格納する。

次の `do` 文で $w2(i1, i2)$ を $w2(i1, i2) * (a(is+i2:$

$n, i s + i 2 - 1) t * a (i s + i 2 : n, i s + i 1 - 1))$ で更新する。
更に次のdo文で、 $w 2 (i 1, i 2)$ を $w 2 (i 1, i 2) + w 2 (i 1, i 1 + 1 : i 2 - 1) * w 2 (i 1 + 1 : i 2 - 1, i 2)$ で更新する。

【0063】

更に、次のdo文で、 $w 2 (i 1, i 2)$ を $w 2 (i 1, i 2) * w 2 (i 2, i 2)$ で更新し、 $w (1 : b l k, 1 : i w i d t h)$ 、 $e v (i s + n : n, 1 : i w i d t h)$ 、 $e v (i e + 1 : i s, 1 : i w i d t h)$ を更新して、処理を元のルーチンに返す。

【0064】

図19～図29は、疑似コードの処理をフローチャートで示した図である。

図19は、実対称行列を3重対角化するサブルーチンtridのフローである。ステップS10においては、サブルーチンとしてshared配列A(k、n)、diag(n)およびsdiag(n)を入力。diag、sdiagは計算した3重対角行列の対角要素、副対角要素を出力として返却する。作業域U(n+1、iblk)、v(n+1、iblk)をルーチン内部で確保しshared属性で利用する。ステップS11において、スレッド生成し、各スレッドでローカル域numthrに総スレッド数nothrdに各スレッドに割り振られたスレッド番号を設定し、各スレッドで以下を設定する。iblkにブロック幅を設定し、 $n b = (n - 2 + i b l k - 1) / i b l k$ 、 $n b a s e = 0$ 、 $i = 1$ を設定する。ステップS12においては、 $i > n b - 1$ か否かを判断する。ステップS12の判断がYESの場合には、ステップS19に進む。ステップS12における判断がNOの場合には、ステップS13において、 $n b a s e = (i - 1) \times i b l k$ 、 $i s t a r t = 1$ 、 $n w i d t h = i b l k$ を設定する。ステップS14において、サブルーチンcopyを呼んで上三角部分に下三角部分をコピーする。ステップS15において、作業領域Uにブロック3重対角化の対象領域をコピーする。すなわち、 $U (n b a s e + 1 : n, 1 : i b l k) \leftarrow A (n b a s e + 1 : n, n b a s e + 1 : n b a s e + i b l k)$ を実行する。ステップS16において、サブルーチンblktridを呼んでUにコピーした部分の3重対角化を行う。(istart=1、ブロック幅はiblkを受け渡す)。ステップS17において、3重対角化されたものを配列Aに戻す。すなわち、 $A (n b a s e + 1 : n, n b a s e + 1 : n b a s e + i b l k) \leftarrow U (n b a s e + 1 : n, 1 : i b l k)$ を行う。ステップS18において、サブルーチンupdateを呼ん

で、 $A(nbase+iblk:n, nbase+iblk:n)$ の下三角部分を更新し、ステップ S 1 2 に戻る。

【 0 0 6 5 】

ステップ S 1 9 においては、 $nbase=(nb-1) \times iblk$ 、 $istart=1$ 、 $iblk2=n-nbase$ と設定する。ステップ S 2 0 においては、作業領域 U にブロック三重対角化の対象領域をコピーする。すなわち、 $U(nbase+1:n, 1:nwidth) \leftarrow A(nbase+1:n, nbase+1:n)$ を行う。ステップ S 2 1 において、サブルーチン blktrid を呼んで U にコピーした部分の三重対角化を行う ($istart=1$ 、ブロック幅は iblk2 を受け渡す)。ステップ S 2 2 において、三重対角化されたものを配列 A に戻す。すなわち、 $A(nbase+1:n, nbase+1:n) \leftarrow U(nbase+1:n, 1:nwidth)$ を行う。ステップ S 2 3 においては、並列処理のために生成したスレッドを消して、サブルーチンを終了する。

【 0 0 6 6 】

図 2 0 は、サブルーチン blktrid のフローである。このサブルーチンは再帰的プログラムとなっている。

このサブルーチンは以下のステートメントによって呼び出される。

```
subroutine blktrid (A,k,n,dig,sdig,nbase,istart,nwidth, U, V, nothrd, n  
umthrd)
```

ここで、nbase はブロックの位置を示すオフセット。istart は再帰呼び出しで対称となる縮小されたブロックのブロック内でのオフセットで最初は 1 で再帰的に呼び出されるとき対象ブロックの位置を示す。nwidth はブロック幅を表す。ステップ S 2 5 においては、 $nwidth < 10$ か否かを判断する。ステップ S 2 5 の判断が NO の場合には、ステップ S 2 7 に進む。ステップ S 2 5 の判断が YES の場合には、ステップ S 2 6 において、サブルーチン btunit を呼んで 3 重対角化を行い、サブルーチンを抜ける。ステップ S 2 7 では、再帰呼び出しのために対象となる更新位置とブロック幅を変えて、呼び出す $istart2=istart$ 、 $nwidth2=nwidth/2$ を設定し受け渡す。縮小されたブロックの開始位置、ブロック幅を受け渡す。ステップ S 2 8 においては、サブルーチン blktrid を再帰的に呼び出す。ステップ S 2 9 においては、スレッド間でバリア同期を取る。ステップ S 3 0 において

は、更新で各スレッドが分担する始点 (is2、is3)、終点 (ie2、ie3) を計算する。すなわち、 $istart3 = istart + nwidth/2$ 、 $nwidth3 = nwidth - nwidth/2$ 、 $is2 = istart2$ 、 $ie2 = istart + nwidth2 - 1$ 、 $is3 = istart3$ 、 $ie3 = istart3 + nwidth3 - 1$ 、 $iptr = nbase + istart3$ 、 $len = (n - iptr + numthrd - 1) / numthrd$ 、 $is = iptr + (nothrd - 1) \times len + 1$ 、 $ie = \min(n, iptr + nothrd \times len)$ を計算する。ステップ S 3 1 においては、 $U(is:ie, is3:ie3) = U(is:ie, is3:ie3) - U(is:ie, is2:ie2) \times W(is3:ie3, is2:ie2)^t - W(is:ie, is2:ie2) \times U(is3:ie3, is2:ie2)^t$ を計算し、ステップ S 3 2 において、スレッド間でバリア同期を取り、ステップ S 3 3 においてサブルーチン blktrid を再帰的に呼び出して、サブルーチンを抜ける。

【 0 0 6 7 】

図 2 1 及び図 2 2 は、サブルーチン blktrid の内部ルーチンであるサブルーチン btunit のフローである。

ステップ S 3 5 において、tmp (numthrd)、sigma、alpha を shared 属性で割り付ける。ステップ S 3 6 において、 $nbase + istart > n - 2$ か否かを判断する。ステップ S 3 6 の判断が YES の場合には、サブルーチンを抜ける。ステップ S 3 6 の判断が NO の場合にはステップ S 3 8 に進む。ステップ S 3 8 では、 $i = istart$ とし、ステップ S 3 9 において、 $i \leq istart - 1 + nwidth$ であるか否かを判断する。ステップ S 3 9 の判断が NO の場合には、サブルーチンを抜ける。ステップ S 3 9 の判断が YES の場合には、ステップ S 4 0 において、各スレッドで分担する始点 is、終点 ie を計算する。 $iptr2 = nbase + i$ 、 $len = (n - iptr2 + numthrd - 1) / numthrd$ 、 $is = iptr2 + (nothrd - 1) \times len + 1$ 、 $ie = \min(n, iptr2 + nothrd \times len)$ を計算する。ステップ S 4 1 では、バリア同期を取り、ステップ S 4 2 では、 $tmp(nothrd) = U(is:ie, i)^t \times U(is:ie, i)$ を計算し、ステップ S 4 3 で、バリア同期を取る。ステップ S 4 4 において、 $nothrd = 1$ か否かを判断する。ステップ S 4 4 の判断が NO の場合には、ステップ S 4 6 に進む。ステップ S 4 4 の判断が YES の場合には、ステップ S 4 5 において、各スレッドで部分計算した和の平方根をとり、三重対角化のための計算を行う (ハウスホルダーベクトルの作成)。 $\sigma = \sqrt{\text{sum}(tmp(1:numthrd))}$ 、ここで、SUM は和、sqrt は平方根である。 $diag(iptr2) = u(iptr2, i)$ 、 $sdiag(iptr2) = -\sigma$ 、 $U(nbase + i + 1, i) = U(nbase$

+i+1,i) +sign(u(nbase+i+1,i)×sigma、alpha=1.0/(sigma×u(nbase+i+1,i) 、
 $U(iptr2,i) = \alpha$ を計算して、ステップS 46に進む。ステップS 46では、
 バリア同期を取り、ステップS 47では、 $iptr3 = iptr2 + 1$ 、ステップS 48では、
 $V(is:ie,i) = A(iptr3:n, iptr2+is:iptr2+ie)^t U(ptr3:n,i)$ を演算し、
 ステップS 49では、バリア同期を取る。

【0068】

ステップS 50では、 $V(is:ie,i) = \alpha \times V(is:ie,i) - V(is:ie,1:i-1) \times (U(iptr3:n,1:i-1)^t \times U(iptr3:n,i)) - U(is:ie,1:i-1) \times (V(iptr3:n,1:i-1)^t \times U(iptr3:n,i))$ を計算し、ステップS 51において、バリア同期を取る。ステップS 52においては、 $tmp(nothr) = V(is:ie,i)^t \times U(is:ie,i)$ を計算し、ステップS 53において、バリア同期を取る。ステップS 54においては、 $nothr=1$ か否かを判断する。ステップS 54の判断がNOの場合には、ステップS 56に進み、ステップS 54の判断がYESの場合には、ステップS 55に進む。ステップS 55においては、 $\beta = 0.5 \times \alpha \times \text{sum}(tmp(1:numthr))$ を計算する。ただし、sumはベクトルの和をとる記号である。ステップS 56では、バリア同期を取る。ステップS 57では、 $V(is:ie,i) = V(is:ie,i) - \beta \times U(is:ie,i)$ を計算し、ステップS 58において、バリア同期を取る。ステップS 59においては、 $ptr2 < n-2$ か否かを判断し、判断がYESの場合には、ステップS 60において、 $U(is:ie,i+1) = U(is:ie,i+1) - U(is:ie,istart:i) \times V(i+1,istart:1)^t - V(is:ie,istart:i) \times U(n+1,istart:1)^t$ を計算し、ステップS 39に戻る。ステップS 59の判断がNOの場合には、ステップS 61において、 $U(is:ie,i+1:i+2) = U(is:ie,i+1:i+2) - U(is:ie,istart:i) \times V(i+1:n,istart:i)^t - V(is:ie,istart:i) \times U(n+1:n,istart:i)^t$ を計算して、サブルーチンを抜ける。

【0069】

図23は、サブルーチンupdateのフローである。

ステップS 65において、バリア同期を取り、ステップS 66において、各スレッドでペアを作り、更新を分担する始点、終点を決める。すなわち、 $nbase2 = nbase + iblk$ 、 $len = (n - nbase2 + 2 \times numthr - 1) / (2 \times numthr)$ 、 $isl = nbase2 + (nothr$

rd-1)len+1、iel=min(n,nbase2+nothrd×len)、nbase3=nbase2+2×numthrd×len、isr=nbase3-nothrd×len+1、ier=min(n,isr+len-1)を計算する。ステップS 6 7では、 $A(iel+1:n, is1:iel) = A(iel+1:n, is1+1:n, is1:iel) - W(iel+1:n, 1:blk) \times U(is1:iel, 1:blk)^t - U(iel+1:n, 1:blk) \times W(is1:iel, 1:blk)^t$ 、 $A(ier+1:n, isr:ier) = A(ier+1:n, isr:ier) - W(ier+1:n, 1:blk) \times U(isr:ier, 1:blk)^t - U(ier+1:n, 1:blk) \times W(isr:ier, 1:blk)^t$ を計算し、ステップS 6 8で、サブルーチンtrupdateを呼んで左半分の対角行列部分を更新する。is1、iel、A、W、Uを渡す。ステップS 6 9において、サブルーチンtrupdateを呼んで右半分の対角行列部分を更新する。isr、ier、A、W、Uを渡す。ステップS 7 0において、バリア同期を取って、サブルーチンを抜ける。

【0 0 7 0】

図2 4は、サブルーチンtrupdate（対角行列部分の更新）のフローである。

更新の始点is、と終点ieを入力とする。これは、サブルーチンが呼び出される前に対角ブロックの下の長方形の部分の更新に使われた始点と終点である。

【0 0 7 1】

ステップS 7 5においてblk2に対角ブロック更新用ブロック幅を設定し、i=isを設定する。ステップS 7 6において、i>ie-1か否かを判断する。ステップS 7 6における判断がYESの場合には、サブルーチンを抜ける。ステップS 7 6における判断がNOの場合には、ステップS 7 7において、各スレッドでの更新の始点、終点を決める。すなわち、is2=i、ie2=min(i+blk2-1, ie-1)、 $A(is2:ie-1, is2, ie2) = A(is2:ie-1, is2, ie2) - U(is2:ie-1, 1:blk) \times W(is2:ie2, 1:blk)^t - W(is2:ie-1, 1:blk) \times U(is2:ie2, 1:blk)^t$ を計算し、ステップS 7 8において、i=i+blk2と設定し、ステップS 7 6に戻る。

【0 0 7 2】

図2 5は、サブルーチンcopyのフローである。

ステップS 8 0において、コピーを各スレッドで対を作って並列実行するときの始点、と幅を計算する。すなわち、 $len = (n - nbase + 2 \times numthrd - 1) / (2 \times numthrd)$ 、 $is1 = nbase + (nothrd - 1) \times len + 1$ 、 $len1 = \max(0, \min(n - is1 + 1, len))$ 、nba

se3=nbase+2×numthrd×len、isr=nbase3-nothrd×len+1、lenr=max (0,min(n-isr+1,len)) を計算する。ステップ S 8 1 において、サブルーチンbandcpを呼び出す。ペアの左側の始点isl、幅lenlできまる領域をコピーする。ステップ S 8 2 において、サブルーチンbandcpを呼び出す。ペアの右側の始点isr、幅lenrできまる領域をコピーする。

【 0 0 7 3 】

図 2 6 は、サブルーチンbandcpのフローである。

このルーチンは、小さな作業域WXを利用してキャッシュ上での転置を行いながらコピーを行う。作業域をWX (nb、nb) とし、始点をis、幅をlenで受ける。

【 0 0 7 4 】

ステップ S 8 5 において、nn=min(nb,len) 、loopx=(len+nn-1) /nn、j=1を演算する。ステップ S 8 6 において、j>loopxか否かを判断する。ステップ S 8 6 の判断がY E S の時は、サブルーチンを終了する。ステップ S 8 6 の判断がN O の場合には、ステップ S 8 7 において、WXにコピーする対角ブロックの大きさnnxとそのオフセットipを決める。ip=is+(j-1)×nn、nl=len-(j-1)×nn、nnx=min(nn,nl)、len2=n-ip-nnx+1、loopy=(len2+nn-1)/nn、TRL (WX (1:nnx,1:nnx)) =TRL (A (ip:ip+nnx-1,ip:ip+nnx-1)) 、TRU (A (ip:ip+nnx-1,ip:ip+nnx-1)) =TRL (WX(1:nnx,i:nnx)) 、i=1, is2=ip, is3=ip+nnxを計算する。ただし、TRUは上三角部分、TRLは下三角部分。

【 0 0 7 5 】

ステップ S 8 8 では、i>loopy-1か否かを判断する。ステップ S 8 8 の判断がN O の場合には、ステップ S 8 9 において、nn×nnxの部分を転置してコピーする。すなわち、WX (1:nn, 1:nnx) =A (is3:is3+nn-1, is2:is2+nnx-1) 、A (is2:is2+nnx-1, is3:is3+nn-1) =WX (1, nn:1, nnx) ^t、is3=is3+nnを計算して、ステップ S 8 8 に戻る。ステップ S 8 8 の判断がY E S の場合には、ステップ S 9 0 で、最後の部分をコピーする。すなわち、nn=n-is3+1、WX (1:nn, 1:nx) =A (is3:n, is2:is2+nnx-1) 、A (is2:is2+nnx-1, is3:n) =WX (1:nn, 1:nx) を演算して、ステップ S 8 6 に戻る。

【0076】

図27は、サブルーチンconvexのフローである。

このルーチンでは、求める固有ベクトルの数nev、ハウスホルダーベクトルはaの下半分に格納されている。3重対角行列の固有ベクトルがev(k, nev)に格納されている。

【0077】

ステップS95において、スレッドを生成する。各スレッドのローカル域のnumthr及びnothrに総スレッド数とスレッド番号(1～numthr)を設定する。ステップS96において、バリア同期を取る。ステップS97において、各スレッドで分担計算する始点、終点を決める。すなわち、 $len = (nev + numthr - 1) / numthr$ 、 $is = (nothr - 1) \times len + 1$ 、 $ie = \min(nev, nothr \times len)$ 、 $width = ie - is + 1$ を計算する。ステップS98において、サブルーチンconvexthrを呼び出し、三重対角行列からもとの行列の固有ベクトルに変換する。各スレッドが担当する固有ベクトルの格納されている領域と固有ベクトルの数widthを渡す。ステップS99において、バリア同期をとり、ステップS100において、生成したスレッドを消滅させ、サブルーチンを抜ける。

【0078】

図28及び図29は、サブルーチンconvexthrのフローである。

このルーチンは、各スレッドごとに分担した三重対角行列の固有ベクトルを変換して、もとの行列の固有ベクトルにする。ハウスホルダ変換を復元するベクトルと係数は配列Aに格納されている。

【0079】

ステップS110において、blkにブロック幅を設定する。ブロック幅は80程度である。ステップS111では、iwidth<0か否かを判断する。ステップS111の判断がYESの場合には、サブルーチンを抜ける。ステップS111の判断がNOの場合には、ステップS112に進む。ステップS112では、以下のループで変換の最初のブロックは、 $(1 + \alpha u u^t)$ を順次計算して求める。まず、 $numblk = (n - 2 + blk - 1) / blk$ 、 $nfbs = n - 2 - blk \times (numblk - 1)$ を求める。ステップS113において、 $i < n - 2 - nfbs + 1$ か否かを判断する。ステップS113の判断が

YES の場合には、ステップ S 114 に進み、 $\alpha = -a(i, i)$ 、 $x(1:iwidth) = a(i+1:n, i)^t \times ev(i+1:n, 1:width)$ 、 $ev(i+1:n, 1:width) = ev(i+1:n, 1:width) + \alpha \times a(i+1:n, i) \times x(1:iwidth)^t$ を演算し、ステップ S 113 に戻る。ステップ S 113 の判断が NO の場合には、ステップ S 115 において、 $i=1$ に設定し、ステップ S 116 において、 $i > numblk-1$ か否かを判断する。ステップ S 116 の判断が NO の場合には、サブルーチンを抜ける。ステップ S 116 の判断が YES の場合には、ステップ S 117 において、ブロック形式の $(1 + U B U^t)$ の $U^t \times E V$ を U^t の左端の上三角行列部分と右側の長方形の部分に分けて計算する。すなわち、 $is = n - 2 - (nfns + i \times blk) + 1$ 、 $ie = ie + blk - 1$ 、 $W(1:blk, iwidth) = a(ie+1:n, is:ie)^t \times ev(ie+1:n, 1:iwidth)$ 、 $W(1:blk-1, 1:iwidth) = w(1:blk-1, 1:iwidth) + TRL(a(is+1:ie, is:ie-1))^t \times ev(is+1:ie, 1:iwidth)$ を計算する。次に、ブロック形式の $(1 + U B U^t)$ の B を求める。 $diag(w2) = -diag(a(is:is+blk-1, is:is+blk-1))$ 、 $i2 = blk$ 、を計算する。 $w2$ の部分に対応する係数 α を格納する。なお、 $TRL(w2)$ は $w2$ の下三角行列、 $diag(x)$ は x の対角要素部分である。

【0080】

ステップ S 118 においては、 $i2 < 1$ か否かを判断する。ステップ S 118 の判断が NO の場合には、ステップ S 119 において、 $w2$ の上三角部分にハウスホルダベクトルの内積 $\times \alpha$ を格納し、 $i1 = i2 - 1$ と設定する。ステップ S 120 では、 $i1 < 1$ か否かを判断し、NO の場合には、ステップ S 121 において、 $w2(i1, i2) = w2(i1, i1) \times (a(is+i2:n, is+i2-1)^t \times a(is+i2:n, is+i1-1))$ 、 $i1 = i1 - 1$ を計算し、ステップ S 120 に戻る。ステップ S 120 の判断が YES の場合には、ステップ S 122 において、 $i2 = i2 - 1$ と設定し、ステップ S 118 に戻る。ステップ S 118 の判断が YES の場合には、ステップ S 123 において、 $i1 = blk - 2$ とし、以下、2 重ループで展開係数を計算する。三角行列の上辺を右から左に決定し、積み上げるように計算する。これは左からハウスホルダ変換をかけたときの展開を足して係数を決めることに相当する。ステップ S 124 において、 $i1 < 1$ か否かを判断する。ステップ S 124 の判断が NO の場合には、ステップ S 125 においては、 $i2 = blk$ と設定し、ステップ S 126 においては、 $i2 < i1 + 1$ か否かを判断する。ステップ S 126 の判断が NO の場合には、ステップ S 127 において、

上辺の要素を左から右に決めていく。一つ前の係数を利用する。すなわち、 $w2(i1, i2) = w2(i1, i2) + w2(i1, i1+1:i2-1) \times w2(i1+1:i2-1, i2)$ 、 $i2 = i2-1$ を計算し、ステップ S 126に戻る。ステップ S 126の判断がYESの場合には、ステップ S 128において、 $i1 = i1-1$ と設定し、ステップ S 124に戻る。ステップ S 124の判断がYESの場合には、ステップ S 129に進み、 $i2 = blk$ を設定し、ステップ S 130において、 $i2 < 1$ か否かを判断する。ステップ S 130の判断がNOの場合には、ステップ S 131において、以下のループで不足していた係数 α をかける。まず、 $i1 = i2-1$ と設定し、ステップ S 132において、 $i1 < 1$ か否かを判断する。ステップ S 132における判断がNOの場合には、ステップ S 133において、 $w2(i1, i2) = w2(i1, i2) \times w2(i2, i2)$ 、 $i1 = i1-1$ を計算し、ステップ S 132に戻る。ステップ S 132の判断がYESの場合には、ステップ S 134において、 $i2 = i2-1$ と設定し、ステップ S 130に戻る。ステップ S 130の判断がYESの場合、ステップ S 135において、 BU^t を計算してWに格納する。 $W(1:blk, 1:iwidth) = TRU(w2) \times W(1:blk, 1:iwidth)$ を計算する。そして、Uの上部の三角行列と下部の長方形の部分とWに格納した BU^t を使って $(1 + UBU^t) \times EV$ を計算する。すなわち、 $ev(ie+1:n, 1:width) = ev(ie+1:n, 1:width) + a(ie+1:n, is:ie) \times W(1:blk, 1:width)$ 、 $ev(is+1:ie, 1:width) = ev(is+1:ie, 1:width) + TRL(a(is+1:ie, is+1:ie)) \times W(1:blk-1, 1:width)$ を計算し、ステップ S 115に戻る。

【0081】

本発明によれば、共有メモリ型スカラ並列計算機を用いて高性能かつスケーラビリティのある固有値・固有ベクトルの並列演算方法を提供できる。

本発明の実施形態によれば、特に、固有ベクトルの変換に関する計算は従来の方法に比べて10倍程度スカラマシンで高速。これらのアルゴリズムを使った実対称行列の固有値・固有ベクトルは、3重対角行列の固有値・固有ベクトルを同じスツルムの方法と逆反復法で求める方法を利用したもので、SUNの数値計算ライブラリSUN Performance libraryの機能に比べて、7個のCPUで6.7倍高速。SUNの他のルーチン（機能が劣る：固有値・固有ベクトルを選択的には求められない）3重対角行列の固有値・固有ベクトルの計算をdivide & conquer

r法で行うものと比べても 2.3 倍高速。

【0082】

これらのアルゴリズムを使ったエルミート行列の固有値・固有ベクトルは、3重対角行列の固有値・固有ベクトルを同じスツルムの方法と逆反復法で求める方法を利用したもので、SUNの数値計算ライブラリSUN Performance libraryの機能に比べて7個のCPUで4.8倍高速。SUNの他のルーチン（機能が劣る：固有値を選択的には求められない）3重対角行列の固有値・固有ベクトルの計算をdivide & conquer法で行うものと比べても3.8倍高速。

【0083】

なお、行列計算の基本的アルゴリズムに関しては、以下の教科書を参照されたい。

G. H. Golub and C. F. Van Loan "Matrix Computations" third edition The Johns Hopkins University Press, 1996

また、3重対角化の並列計算については、以下の文献を参照されたい。

J. Choi, J. J. Dongarra, and D. W. Walker, "THE DESIGN OF A PARALLEL DENSE LINEAR ALGEBRA SOFTWARE LIBRARY: REDUCTION TO HESSENBERG, TRIDIAGONAL, AND BIDIAGONAL FORM", Engineering Physics and Mathematics Division, Mathematical Sciences Section, prepared by the Oak Ridge National Laboratory managed by Martin Marietta Energy System, Inc., for the U.S. DEPARTMENT OF ENERGY under Contract No. DE-AC05-84OR21400, ORNL/TM-12472.

（付記1）共有メモリ型スカラ並列計算機用固有値問題の並列処理方法を実現させるプログラムであって、

演算すべき実対称行列あるいはエルミート行列をブロック化し、ブロック化した部分をメモリの作業域にコピーして、ブロック間の積を利用して3重対角化するステップと、

3重対角化された行列から固有値と固有ベクトルを求めるステップと、

該3重対角化された行列から求められた固有ベクトルを、所定のブロック幅の行列演算の並列演算となるように変形されたハウスホルダー変換によって変換し、元の行列の固有ベクトルを求めるステップと、

を備えることを特徴とする並列処理方法を共有メモリ型スカラ並列計算機に実現させるプログラム。

【0084】

(付記2) 前記3重対角化ステップにおいて、ブロック化された部分の更新を再帰的プログラムで行うことを特徴とする付記1に記載のプログラム。

(付記3) 前記3重対角化ステップにおいて、ブロック化された部分は、キャッシュメモリのページを跨らないように小さなブロックに分割して格納し、各プロセッサで並列に演算を行うことによって処理されることを特徴とする付記1に記載のプログラム。

【0085】

(付記4) 前記固有ベクトルを求めるステップにおいて、前記ハウスホルダー変換のための行列は、各プロセッサで処理できるような小さな係数行列となる上三角行列を各プロセッサで同時に生成されることを特徴とする付記1に記載のプログラム。

【0086】

(付記5) 前記固有ベクトルを求めるステップにおいて、前記固有ベクトルは、格納されている2次元配列の2次元目をプロセッサの数に対応して均等に分割し、おのおのの分割された領域を各プロセッサに割り当てることによって、演算されることを特徴とする付記1に記載のプログラム。

【0087】

(付記6) 共有メモリ型スカラ並列計算機用固有値問題の並列処理方法であって、

演算すべき実対称行列あるいはエルミート行列をブロック化し、ブロック化した部分をメモリの作業域にコピーして、ブロック間の積を利用して3重対角化するステップと、

3重対角化された行列から固有値と固有ベクトルを求めるステップと、

該3重対角化された行列から求められた固有ベクトルを、所定のブロック幅の行列演算の並列演算となるように変形されたハウスホルダー変換によって変換し、元の行列の固有ベクトルを求めるステップと、

を備えることを特徴とする並列処理方法。

【0088】

(付記7) 前記3重対角化ステップにおいて、ブロック化された部分の更新を再帰的プログラムで行うことを特徴とする付記6に記載の並列処理方法。

(付記8) 前記3重対角化ステップにおいて、ブロック化された部分は、キャッシュメモリのページを跨らないように小さなブロックに分割して格納し、各プロセッサで並列に演算を行うことによって処理されることを特徴とする付記6に記載の並列処理方法。

【0089】

(付記9) 前記固有ベクトルを求めるステップにおいて、前記ハウスホルダー変換のための行列は、各プロセッサで処理できるような小さな係数行列となる上三角行列を各プロセッサで同時に生成されることを特徴とする付記6に記載の並列処理方法。

【0090】

(付記10) 前記固有ベクトルを求めるステップにおいて、前記固有ベクトルは、格納されている2次元配列の2次元目をプロセッサの数に対応して均等に分割し、おのおのの分割された領域を各プロセッサに割り当てることによって、演算されることを特徴とする付記6に記載の並列処理方法。

【0091】

【発明の効果】

高性能かつスケーラビリティのある固有値・固有ベクトルの解法を実現できる。

【図面の簡単な説明】

【図1】

本発明の実施形態が前提とする共有メモリ型スカラ計算機のハードウェア構成を示した図である。

【図2】

本発明の実施形態のアルゴリズムを説明する図(その1)である。

【図3】

本発明の実施形態のアルゴリズムを説明する図（その 2）である。

【図 4】

本発明の実施形態のアルゴリズムを説明する図（その 3）である。

【図 5】

本発明の実施形態のアルゴリズムを説明する図（その 4）である。

【図 6】

本発明の実施形態のアルゴリズムを説明する図（その 5）である。

【図 7】

本発明の実施形態のアルゴリズムを説明する図（その 6）である。

【図 8】

本発明の実施形態のアルゴリズムを説明する図（その 7）である。

【図 9】

本発明の実施形態のアルゴリズムを説明する図（その 8）である。

【図 1 0】

本発明の実施形態のアルゴリズムを説明する図（その 9）である。

【図 1 1】

本発明の実施形態のアルゴリズムを説明する図（その 1 0）である。

【図 1 2】

本発明の実施形態に従ったルーチンの疑似コード（その 1）である。

【図 1 3】

本発明の実施形態に従ったルーチンの疑似コード（その 2）である。

【図 1 4】

本発明の実施形態に従ったルーチンの疑似コード（その 3）である。

【図 1 5】

本発明の実施形態に従ったルーチンの疑似コード（その 4）である。

【図 1 6】

本発明の実施形態に従ったルーチンの疑似コード（その 5）である。

【図 1 7】

本発明の実施形態に従ったルーチンの疑似コード（その 6）である。

【図 18】

本発明の実施形態に従ったルーチンの疑似コード（その 7）である。

【図 19】

疑似コードの処理をフローチャートで示した図（その 1）である。

【図 20】

疑似コードの処理をフローチャートで示した図（その 2）である。

【図 21】

疑似コードの処理をフローチャートで示した図（その 3）である。

【図 22】

疑似コードの処理をフローチャートで示した図（その 4）である。

【図 23】

疑似コードの処理をフローチャートで示した図（その 5）である。

【図 24】

疑似コードの処理をフローチャートで示した図（その 6）である。

【図 25】

疑似コードの処理をフローチャートで示した図（その 7）である。

【図 26】

疑似コードの処理をフローチャートで示した図（その 8）である。

【図 27】

疑似コードの処理をフローチャートで示した図（その 9）である。

【図 28】

疑似コードの処理をフローチャートで示した図（その 10）である。

【図 29】

疑似コードの処理をフローチャートで示した図（その 11）である。

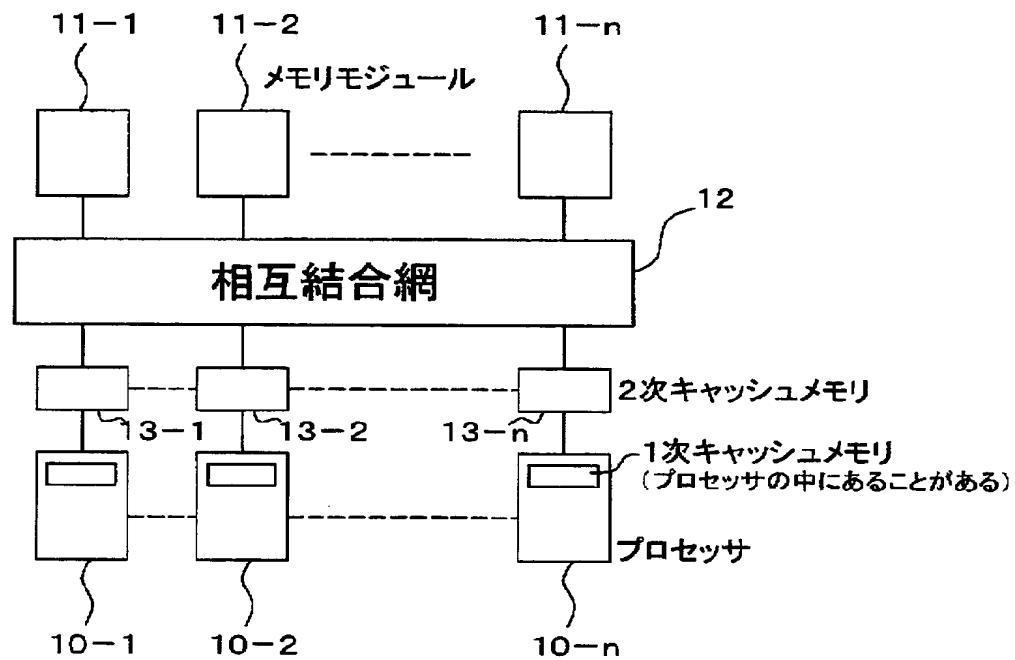
【符号の説明】

10-1～10-n	プロセッサ
11-1～11-n	メモリモジュール
12	相互結合網
13-1～13-n	2次キャッシュメモリ

【書類名】 図面

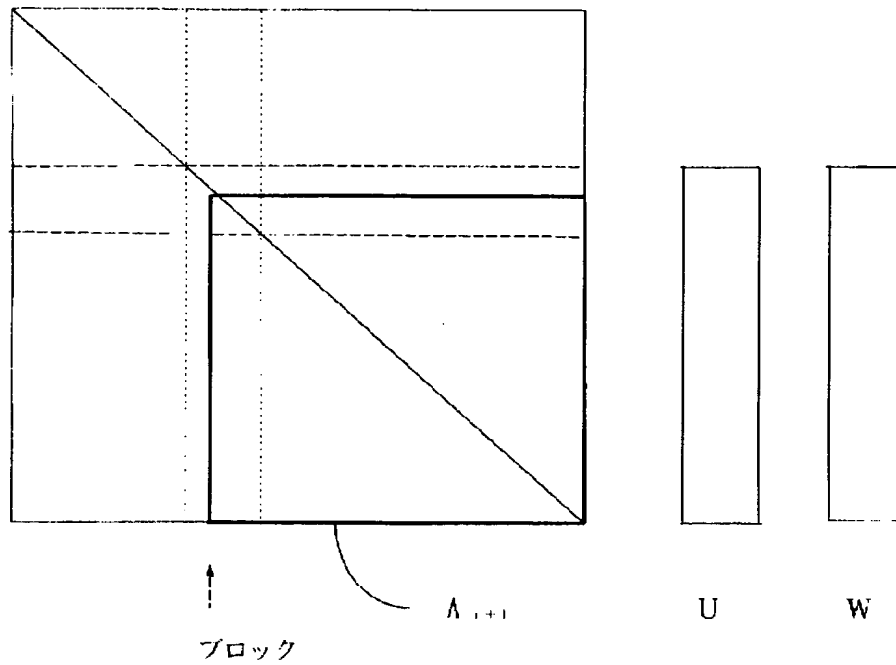
【図 1】

本発明の実施形態が前提とする共有メモリ型
スカラ計算機のハードウェア構成を示した図



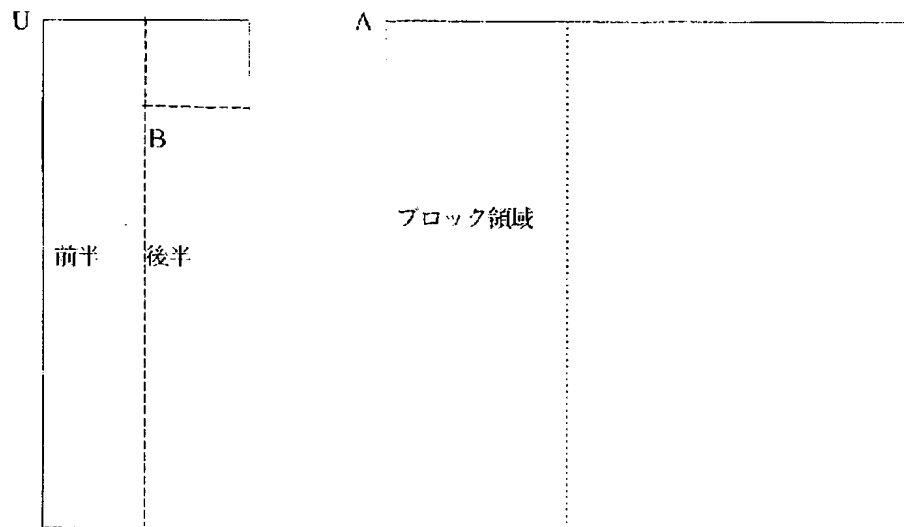
【図 2】

本発明の実施形態のアルゴリズムを説明する図（その 1）



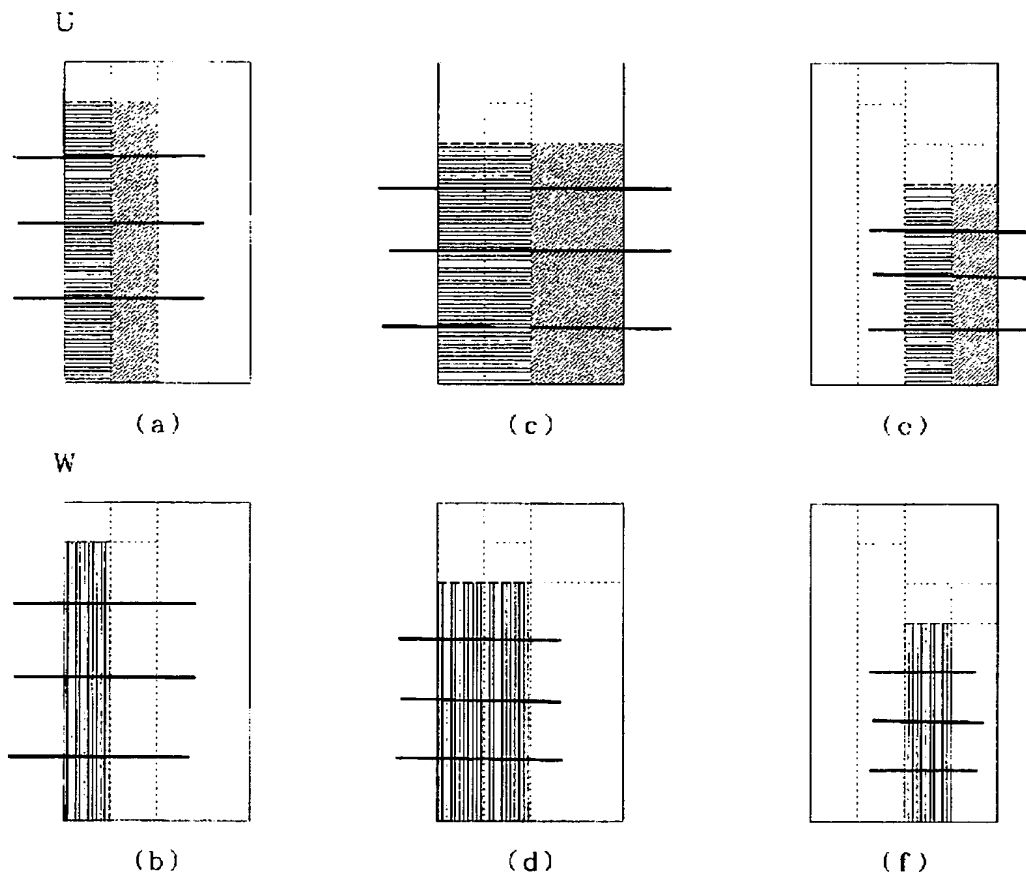
【図 3】

本発明の実施形態のアルゴリズムを説明する図（その 2）



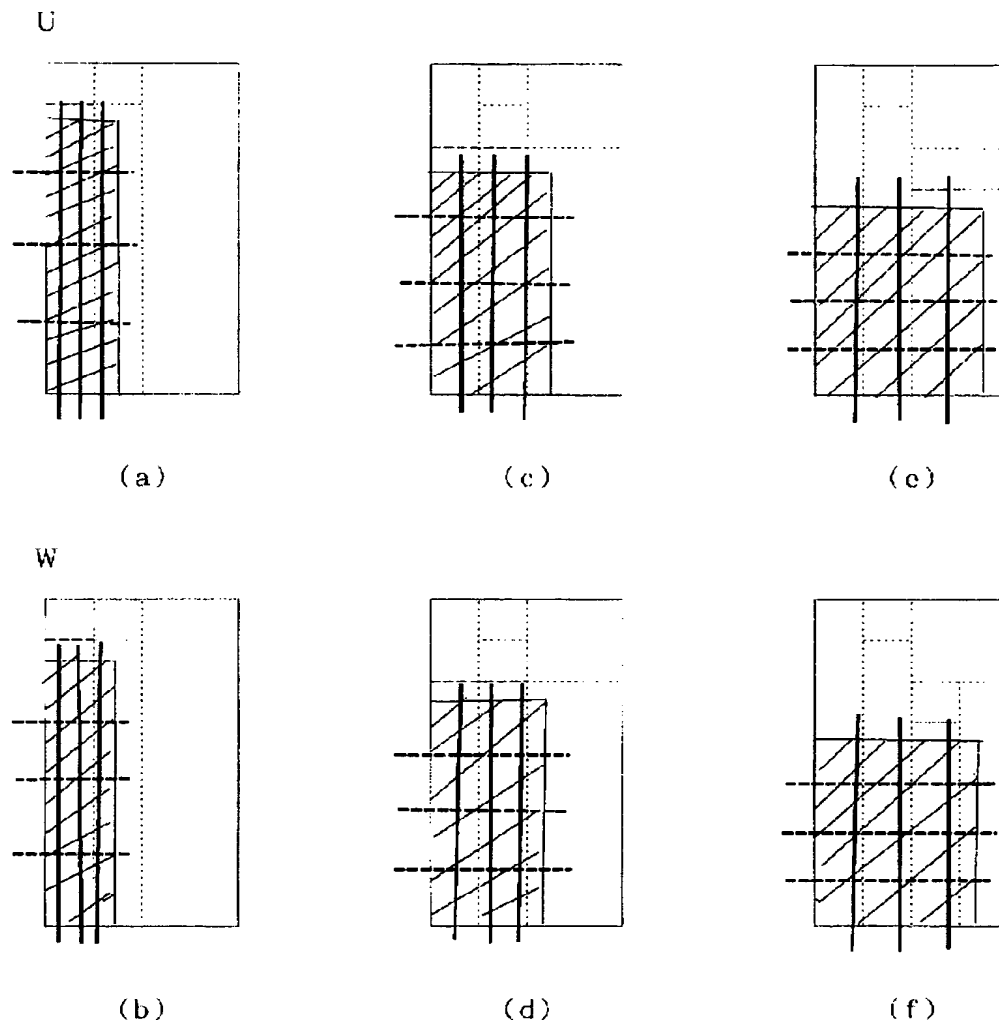
【図 4】

本発明の実施形態のアルゴリズムを説明する図（その 3）



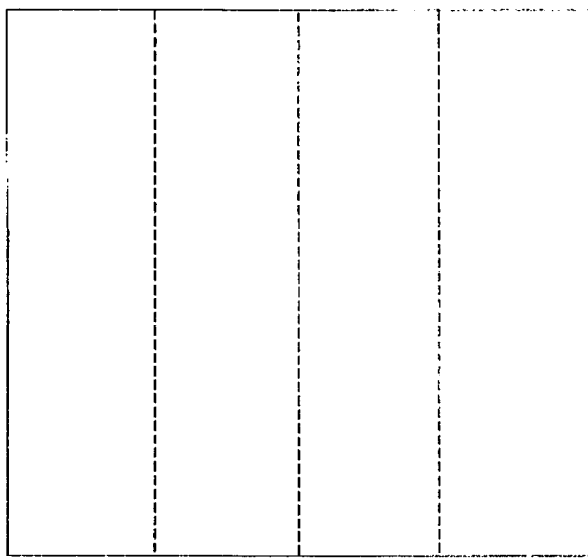
【図 5】

本発明の実施形態のアルゴリズムを説明する図（その 4）



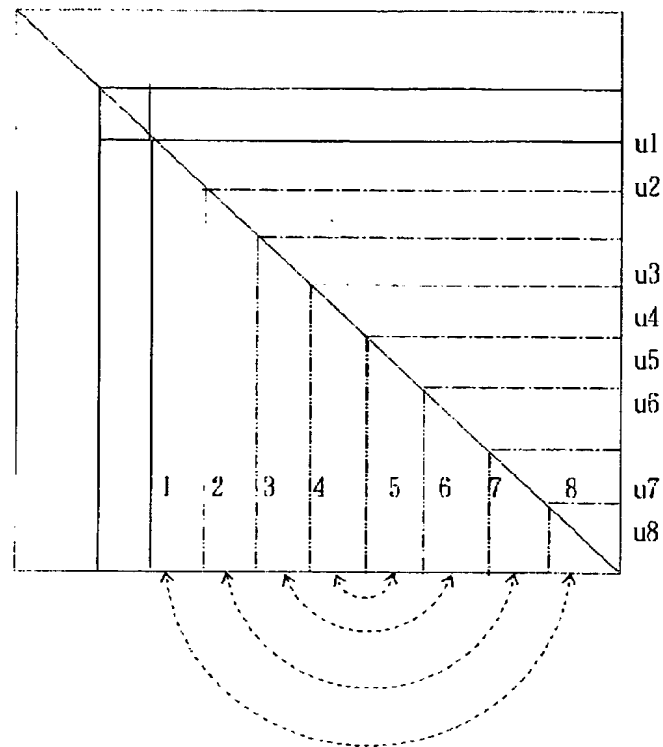
【図 6】

本発明の実施形態のアルゴリズムを説明する図（その 5）

 Λ_{n+1}

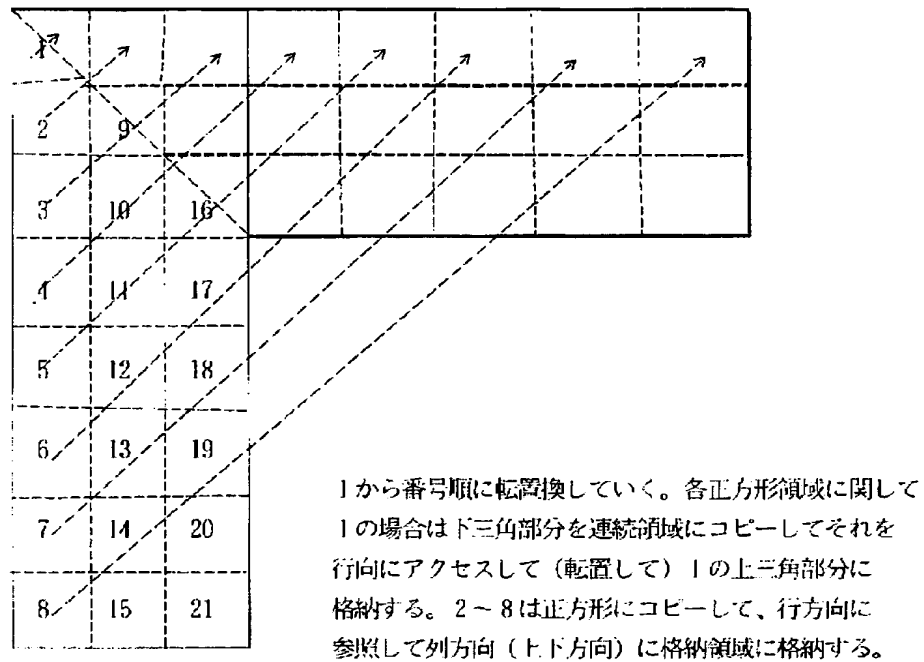
【図 7】

本発明の実施形態のアルゴリズムを説明する図（その 6）



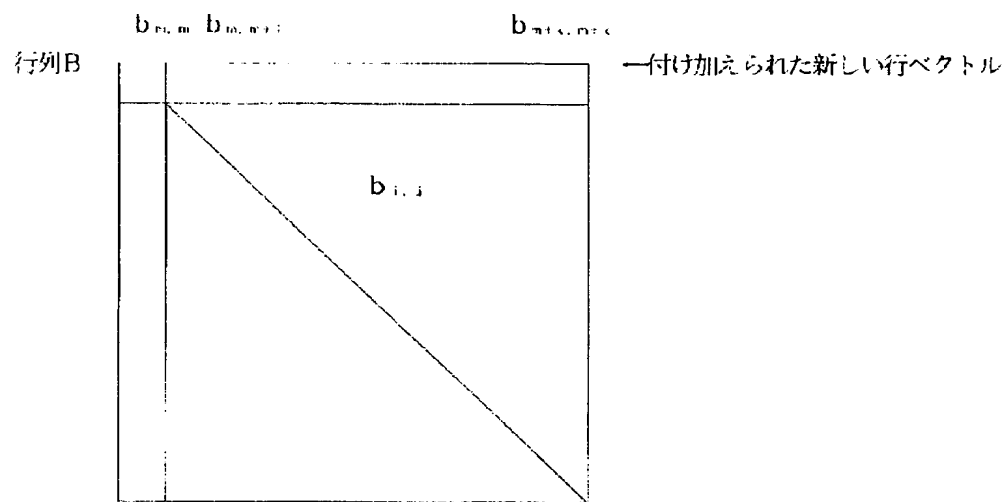
【図 8】

本発明の実施形態のアルゴリズムを説明する図（その 7）



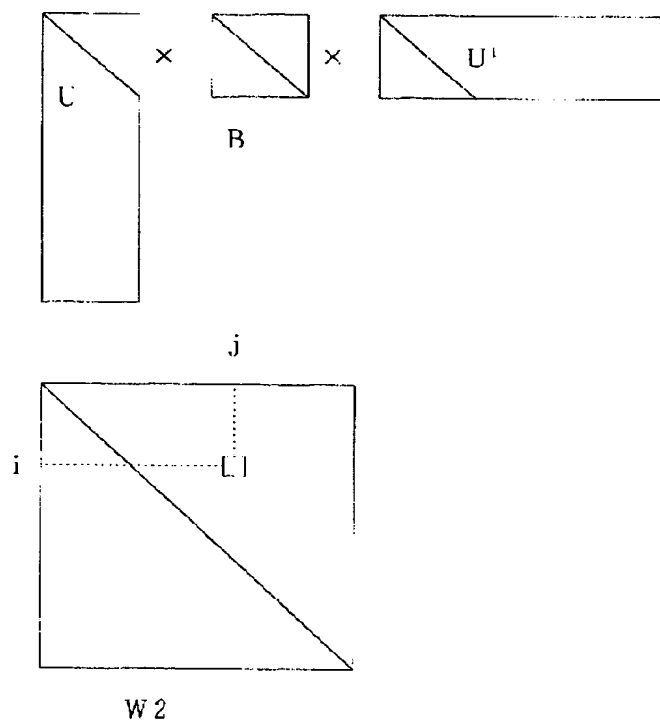
【図 9】

本発明の実施形態のアルゴリズムを説明する図（その 8）



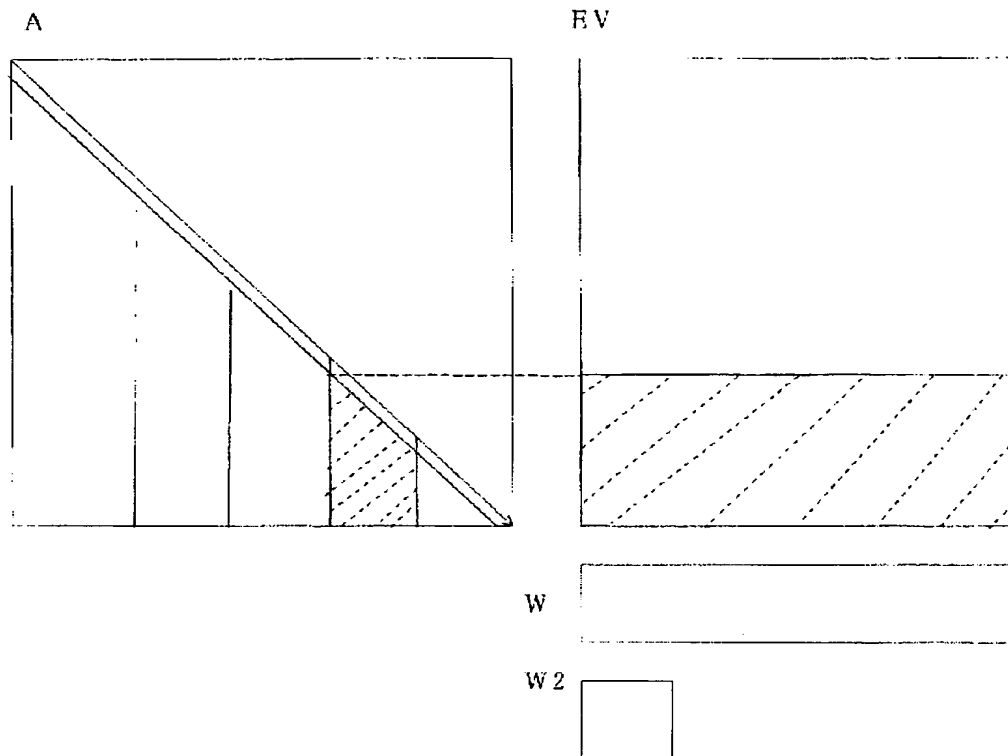
【図 1 0】

本発明の実施形態のアルゴリズムを説明する図（その 9）



【図 11】

本発明の実施形態のアルゴリズムを説明する図（その 10）



【図 12】

本発明の実施形態に従ったルーチンの疑似コード（その 1）

```

c      実対称行列を 3 重対角化するサブルーチン
      subroutine trid(a,k,n,diag,sdiag)
      ! a に実対象行列の下三角部分を格納する。daig,sdiag に三重対角行列の対角、副
      ! 対角部分を格納する。出力として a の下半角部分に変換に必要な情報が格納される。
      constant iblk←'set block width'
      shared array a(k,n),diag(n),sdiag(n)
      allocate shared array u(n+1,iblk),v(n+1,iblk)
      ! U はブロック 3 重化を行うブロックを格納、V は W を格納する領域
c      create threads
      create threads
      set nothrd and numthrd
c      nothrd は各スレッドごとの番号で 1 ～ #TH、numthrd=#TH（スレッドの総数）
      nb=(n-2+iblk-1)/iblk
      nbase=0
      do i=1,nb-1
      nbase=(i-1)*iblk
      istart=1
      nwidth=iblk
      call copy(a,k,n,nbase,nothrd,numthrd)
c      copy
      u(nbase+1:n,nbase+1:nbase+iblk)←a(nbase+1:n,nbase+1:nbase+iblk)
      call blktrid(a,k,n,diag,sdiag,nbase,istart,nwidth,
      u,v,nothrd,numthrd) ! LU 分解を並列に行う
c      copy back
      a(nbase+1:n,nbase+1:nbase+iblk)←u(nbase+1:n,nbase+1:nbase+iblk)
      call update(a,k,n,nbase,nwidth,u,v,nothrd,numthrd)
      enddo
      nbase=(nb-1)*iblk
      istart=1
      nwidth=n-nbase
      call blktrid(a,k,n,diag,sdiag,nbase,istart,nwidth,
      u,v,nothrd,numthrd)
      return
      end

```

【図 13】

本発明の実施形態に従ったルーチンの疑似コード（その2）

```

c   ブロック行列を3重対角化するルーチンで再帰的に呼び出される
c   nbase はブロックの位置を示すオフセット、istart は再帰呼び出しで対象となる
c   縮小されたサブブロックのブロック内でのオフセットで最初1で再帰的に呼び出
c   されるとき対象サブブロックの位置をしめす。nwidth はサブブロックの大きさ。
subroutine blktrid(a,k,n,diag,sdiag,nbase,istart,nwidth,
                  u,v,nothrd,numthrd)
shared array a(k,n),diag(n),sdiag(n),u(n+1,*),v(n+1,*)

if(nwidth<10)then
call btunit(a,k,n,diag,sdiag,nbase,istart,nwidth,
            u,v,nothrd,numthrd)
else
istart2←istart
nwidth2←nwidth/2
call blktrid(a,k,n,diag,sdiag,nbase,istart2,nwidth2,
            u,v,nothrd,numthrd)
BARRIER SYNC
istart3←istart+nwidth/2
nwidth3←nwidth-nwidth/2
is2←istart2
ie2←istart+nwidth2-1
is3←istart3
ie3←istart3+nwidth3-1
iptr←nbase+istart3
len←(n-iptr+numthrd-1)/numthrd
is←iptr+(nothrd-1)*len+1
ie←min(n,iptr+nothrd*len)
u(is:ie,is3:ie3)←u(is:ie,is3:ie3)
               -u(is:ie,is2:ie2)*w(is3:ie3,is2:ie2)t
               -W(is:ie,is2:ie2)*U(is3:ie3,is2:ie2)t
BARRIER SYNC
call blktrid(a,k,n,diag,sdiag,nbase,istart3,nwidth3,
            u,v,nothrd,numthrd)
endif
return
end

```

【図 14】

本発明の実施形態に従ったルーチンの疑似コード (その3)

```

c      blktrid の内部ルーチン
      subroutine btunit(a,k,n,diag,sdiag,nbase,istart,nwidth,
                     u,v,nothrd,numthrd)
      shared :: a(k,n),diag(n),sdiag(n),u(n+1,*),v(n+1,*)
      shared :: tmp(numthrd),sigma,alpha
      if (nbase+istart>n-2) then
      return
      endif

      do i=istart,istart-1+nwidth
      iptr2←nbase+i
      len←(n-iptr2+numthrd-1)/numthrd
      is←iptr2+(nothrd-1)*len+1
      ie←min(n,iptr2+nothrd*len)
      BARRIER SYNC
      tmp(nothrd)←u(is:ie,i)t*u(is:ie,i)
      BARRIER SYNC
      if (nothrd=1) then
      sigma←sqrt(sum(tmp(1:numthrd))) ! SUM は和、sqrt は平方根
      diag(iptr2)←u(iptr2,i)
      sdiag(iptr2)←-sigma
      u(nbase+i+1,i)←u(nbase+i+1,i)+sign(u(nbase+i+1,i)*sigma
      alpha=1.0/(sigma*u(nbase+1+1,i))
      u(iptr2,i)=alpha
      endif
      BARRIER SYNC
      iptr3=iptr2+1
      v(is:ie,i)
      ←A(iptr3:n,iptr2+is:iptr2+ie)t*u(iptr3:n,i)
      BARRIER SYNC
      len2←(i-1+numthrd-1)/numthrd
      isx←(nothrd-1)*len2+1
      iex←min(i-1,nothrd*len2)
      u(n+1,isx:iex)←u(nbase+i+1:n,isx:iex)t*u(i+1:n,i)
      v(n+1,isx:iex)←v(nbase+i+1:n,isx:iex)t*u(i+1:n,i)
      BARRIER SYNC
      v(is:ie,i)←alpha*(v(is:ie,i)-v(is:ie,1:i-1)*u(n+1,1:i-1)t
      -u(is:ie,1:i-1)*v(n+1,1:i-1)t)
      BARRIER SYNC
      tmp(nothrd)←v(is:ie,i)t*u(is:ie,i)
      BARRIER SYNC
      if (nothrd=1) then
      beta←0.5*alpha*sum(tmp(1:numthrd))
      endif
      BARRIER SYNC
      v(is:ie,i)←v(is:ie,i)-beta*u(is:ie,i)
      BARRIER SYNC
      if (i<iblk ) then
      if (ptr2<n-2) then
      u(is:ie,i+1)←u(is:ie,i+1)-u(is:ie,istart:i)*v(i+1,istart:i)t
      -v(is:ie,istart:i)*U(i+1,istart:i)t
      else
      u(is:ie,i+1:i+2)←u(is:ie,i+1:i+2)-u(is:ie,istart:i)*v(n-1:n,istart:i)t
      -v(is:ie,istart:i)*u(n-1:n,istart:i)t
      return
      endif
      endif
      enddo

      eliminate threads

      return
      end

```

【図 15】

本発明の実施形態に従ったルーチンの疑似コード（その 4）

```

c      u,v から行列の下半分を更新するルーチン
c      nbase はブロックの位置を示すオフセット。nwidth はブロック幅
      subroutine update(a,k,n,nbase,nwidth,u,v,nothrd,numthrd)
      shared array a(k,n),u(n+1,*),v(n+1,*)

      BARRIER SYNC
      blk←nwidth
      nbase2←nbase+nwidth
      len←(n-nbase+2*numthrd-1)/(2*numthrd)
      isl←nbase+(nothrd-1)*len+1
      iel←min(n,nbase+nothrd*len)
      nbase3←nbase+2*numthrd*len
      isr←nbase3-nothrd*len+1
      ier←min(n,isr+len-1)
      a(iel+1:n,isl:iel)
      ←a(iel+1:n,isl:iel)-w(iel+1:n,1:blk)*u(isl:iel,1:blk)t
      -u(iel+1:n,1:blk)*w(isl:iel,1:blk)t
      a(ier+1:n,isr:ier)
      ←a(ier+1:n,isr:ier)-w(ier+1:n,1:blk)*u(isr:ier,1:blk)t
      -u(ier+1:n,1:blk)*w(isr:ier,1:blk)t

      call trupdate(a,k,n,isl,iel,u,v,blk)
      call trupdate(a,k,n,isr,ier,u,v,blk)
      BARRIER SYNC
      return
      end

c      対角行列部分の更新
      subroutine trupdate(a,k,n,is,ie,u,v,blk)
      constant blk2←対角ブロック更新用ブロック幅
      shared array a(k,n),u(n+1,*),v(n+1,*)

      do i=is,ie,blk2
      is2←i
      ie2←min(i+blk2-1,ie)
      a(is2:ie,is2:ie2)
      ←a(is2:ie,is2:ie2)-w(is2:ie,1,blk)*u(is2:ie2,1:blk)t
      -u(is2:ie,1,blk)*w(is2:ie2,1:blk)t
      enddo

      return
      end

      subroutine copy(a,k,n,nbase,nothrd,numthrd)
      len←(n-nbase+2*numthrd-1)/(2*numthrd)
      isl←nbase+(nothrd-1)*len+1
      lenl=max(0,min(n-isl+1,len))
      nbase3←nbase+2*numthrd*len
      isr←nbase3-nothrd*len+1
      lenr=max(0,min(n-isr+1,len))

      call bandcp(a,k,n,isl,len)
      call bandcp(a,k,n,isr,ier)

      return
      end

```

【図 16】

本発明の実施形態に従ったルーチンの疑似コード（その5）

o 更新した下三角部分を上三角部分にコピーするルーチン

```

subroutine bandcp(a,k,n,is,len)
constant nb←size of small buffer
private w(nb,nb)

nn←min(nb,len)
loopx←(len+nn-1)/nn

do j=1,loopx
ip←is+(j-1)*nn
n1←len-(j-1)*nn
nnx←min(nn,n1)
len2←n-ip+1
loopy←(len2+nnx-1)/nnx
is2=is+(j-1)*nnx
TRL(w(1:nnx,1:nnx))←TRL(a(is2:is2+nnx-1,is2:is2+nnx))
TRU(a(is2:is2+nnx-1,is:is+nnx))←TRL(w(1:nnx,1:nnx))t
!   TRL は下三角行列部分、TRU は上三角部分

do i=2,loopy-1
is3←is2+(i-1)*nnx
w(1:nnx,1:nnx)←a(is3:is3+nnx-1,is2:is2+nnx)
a(is2:is2+nnx,is3:is3+nnx-1)←w(1,nnx:1,nnx)t
enddo

is3+(loopy-1)*nnx
ny←n-is3+1
w(1:ny,1:nx)←a(is3:n,is2:is2+nnx)
a(is2:is2+nnx,is3:n)←w(1,ny:1,nx)t
enddo

return
end

```

【図 17】

本発明の実施形態に従ったルーチンの疑似コード（その6）

```
c      3重対角行列の固有ベクトル (ev(1:n,1:nev)に格納されている) を元の行列の固有
c      ベクトルに変換するルーチン
c      a は三重対角化の出力で変換に必要な情報を下三角部分に格納
c
      subroutine convey(a,k,n,ev,nev)
      shared array a(k,n),ev(k,n)

c      create threads
c      set nothrd and numthrd
c      nothrd は各スレッドごとの番号で 1 ~ #TH、numthrd=#TH (スレッドの総数)

      BARRIER SYNC
      len←(nev+numthrd-1)/numthrd
      is←(nothrd-1)*len+1
      ie←min(nev,nothrd*len)
      nevthrd←max(ie-is+1,0)
      call conveythrd(a,k,n,ev(1,is),nevthrd)
      BARRIER SYNC

      return
      end
```

【図 18】

本発明の実施形態に従ったルーチンの疑似コード（その 7）

```

c   固有ベクトルを変換するルーチン
subroutine conveythrd(a,k,n,av,iwidth)
  constat blk←ブロック幅
  shared array a(k,n)
  array ev(k,*)
  private w(blk,n),w2(blk,blk)

  if(iwidth<0) then
    return
  endif

  numblk=(n-2+blk-1)/blk
  nfbs←n-2-bk*(numblk-1)
  do i=n-2,n-2-nfbs+1,-1
    alpha←-a(i,i) ! alpha は 3 重対角化時に対角要素に格納される
    x(1:iwidth)←ev(i+1:n,1:iwidth)t*a(i+1:n,i)
    ev(i+1:n,1:iwidth)←
    ev(i+1:n,1:iwidth)+alpha*a(i+1:n,i)*x(1:iwidth)t
  enddo

  do i=1,numblk-1
    is←n-2-(nfbs+i*blk)+1
    ie←is+blk-1
    w(1:blk,iwidth)
    ←a(is+1:n,is:ie)t*ev(is+1:n,1:iwidth)
    w(1:blk-1,1:iwidth)←w(1:blk-1,1:iwidth)
    +TRL(a(ie+1:is,ie:ie))t*ev(ie+1:is,1:iwidth)
    ! TRL は下三角行列部分
    DIAG(w2)←DIAG(a(is:ie,ie:ie)) ! DIAG 行列の対角要素ベクトル
    do i2=blk,1,-1
      do i1=i2-1,1,-1
        w2(i1,i2)=
        ←w2(i1,i1)*(a(is+i2:n,is+i2-1)t*a(is+i2:n,is+i1-1))
      enddo
    enddo
    do i1=blk-1,1,-1
      do i2=blk,i1+1,-1
        w2(i1,i2)←w2(i1,i2)+w2(i1,i1+1:i2-1)*w2(i1+1:i2-1,i2)
      enddo
    enddo

    do i2=blk,1,-1
      do i1=i2-1,1,-1
        w2(i1,i2)←w2(i1,i2)*w2(i2,i2)
      enddo
    enddo

    w(1:blk,1:iwidth)←
    w(1:blk,1:iwidth)+TRU(w2)*w(1:blk,1:iwidth) ! TLU は上三角行列部分

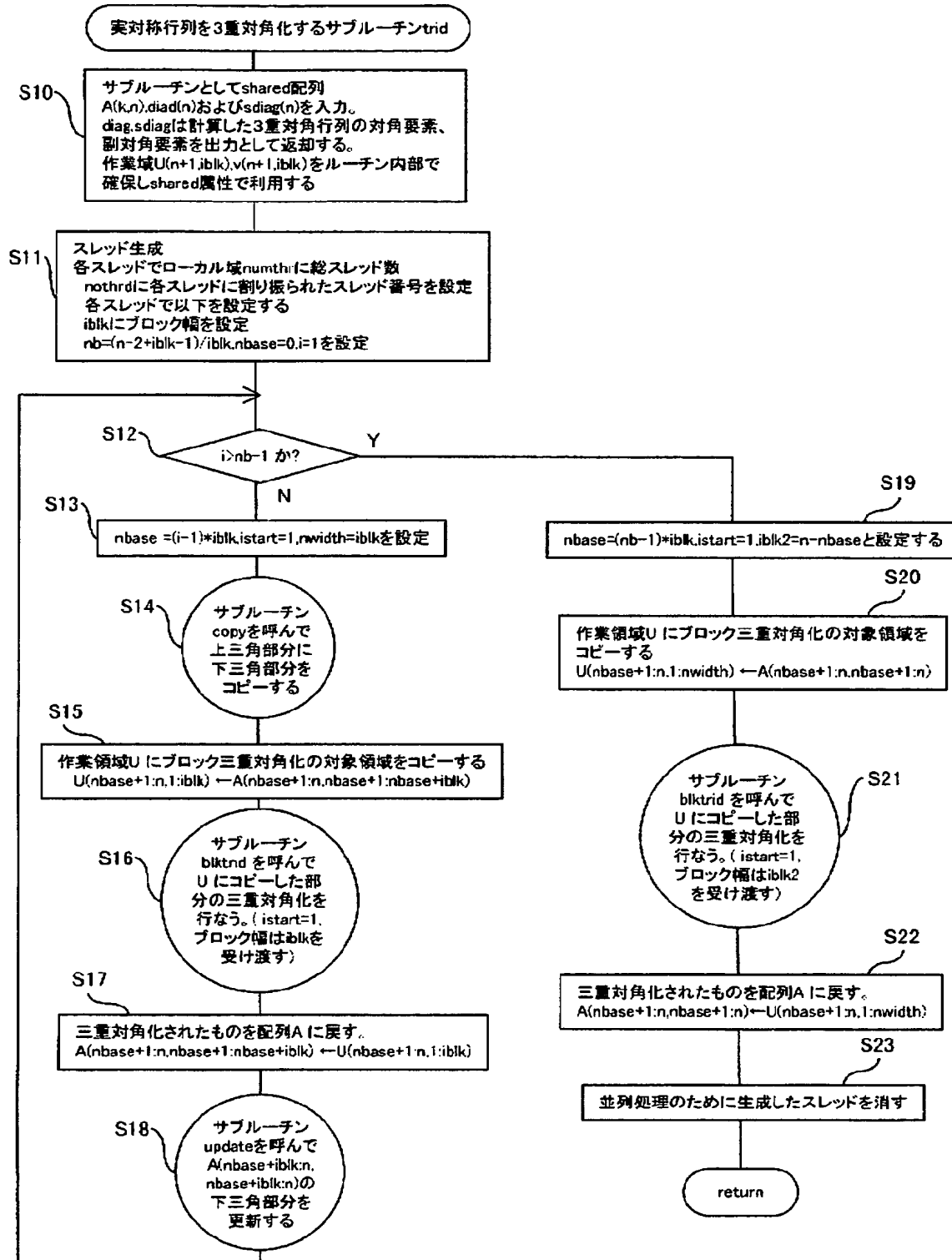
    ev(is+1:n,1:iwidth)
    ←a(is+1:n,is:ie)*w(1:blk,1:iwidth)
    ev(ie+1:is,1:iwidth)
    ←TRL(a(ie+1:is,ie:ie-1))*w(1:blk-1,1:iwidth)
  enddo

  return
end

```

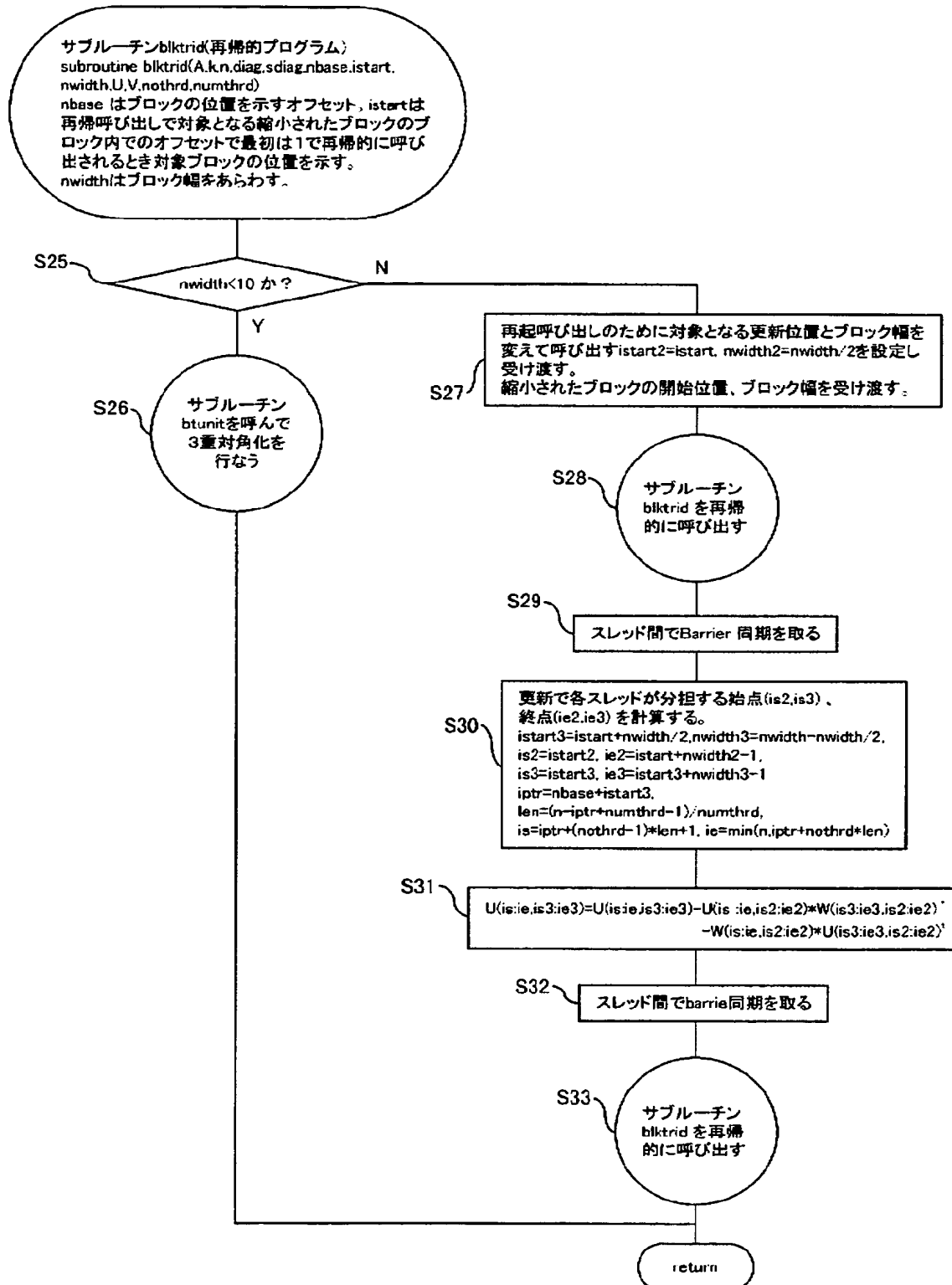
【図 19】

疑似コードの処理をフローチャートで示した図(その1)



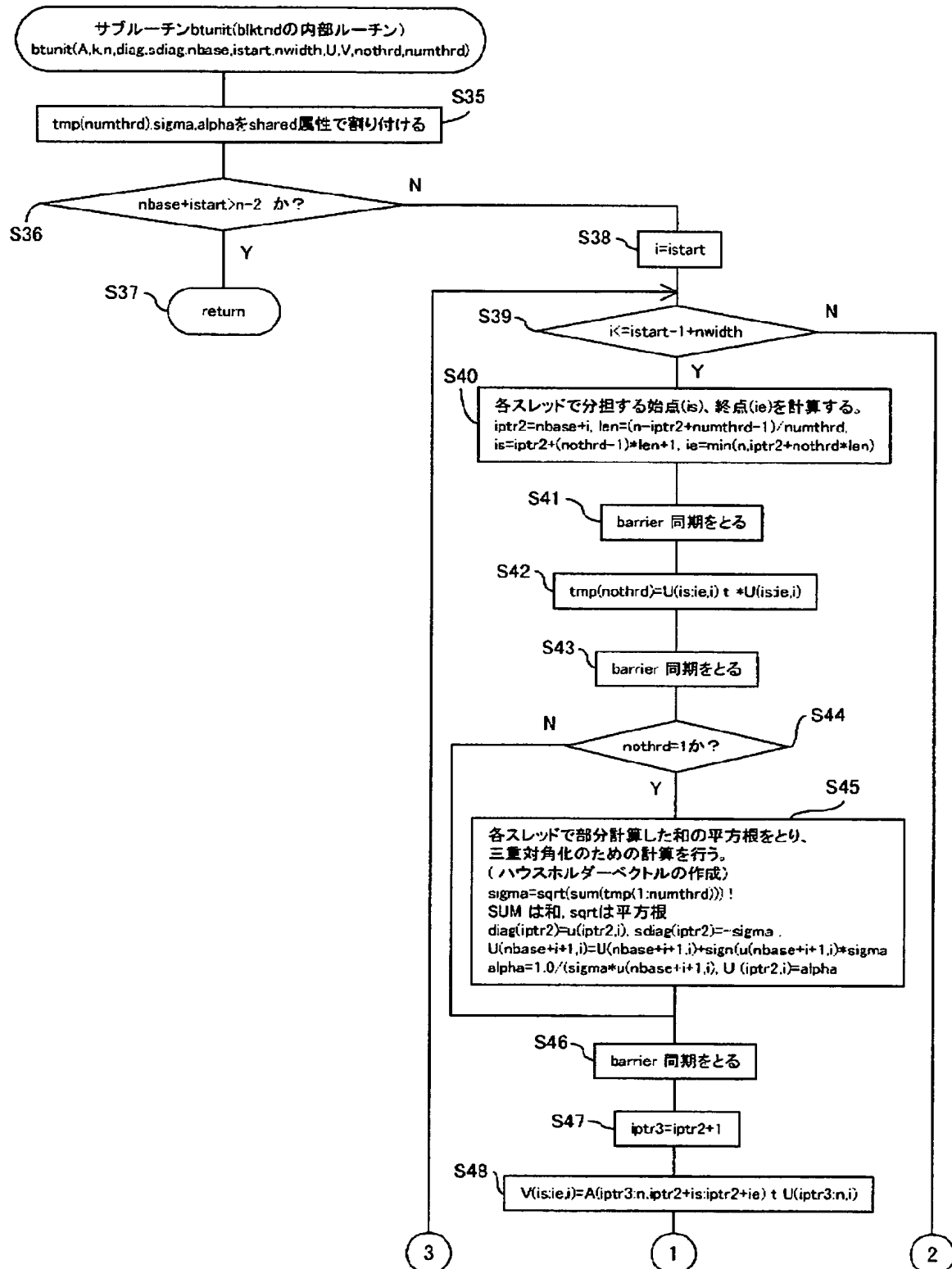
【図 20】

疑似コードの処理をフローチャートで示した図(その2)



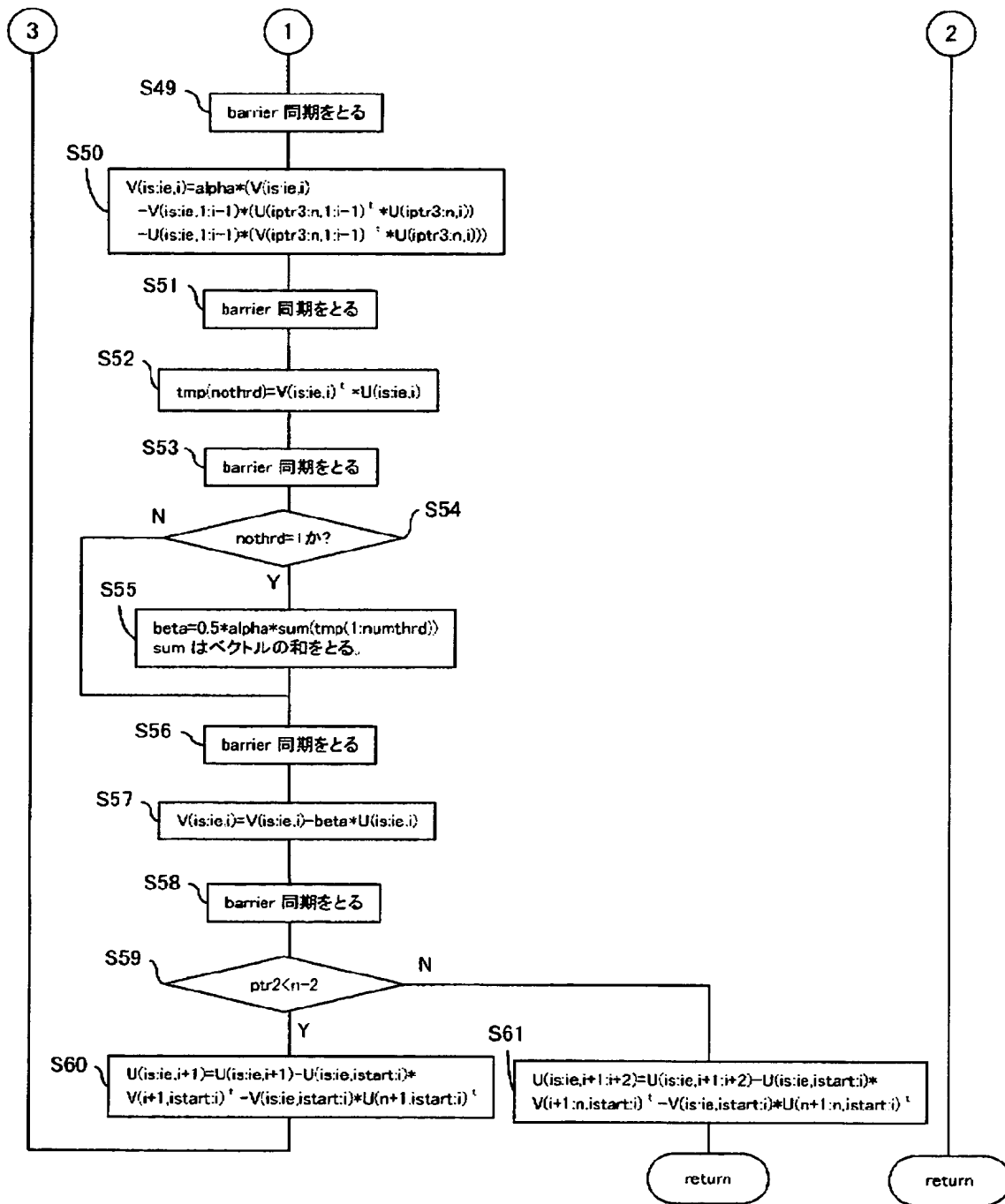
【図 21】

疑似コードの処理をフローチャートで示した図(その3)



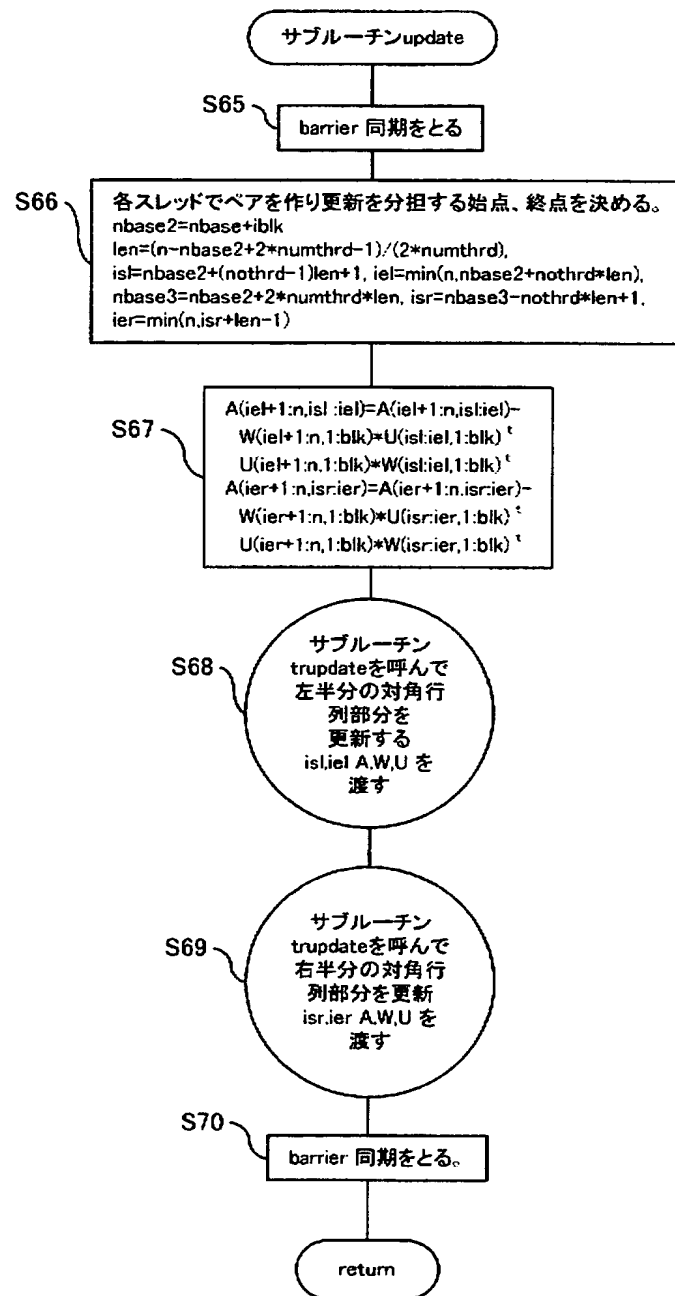
【図 22】

疑似コードの処理をフローチャートで示した図(その4)



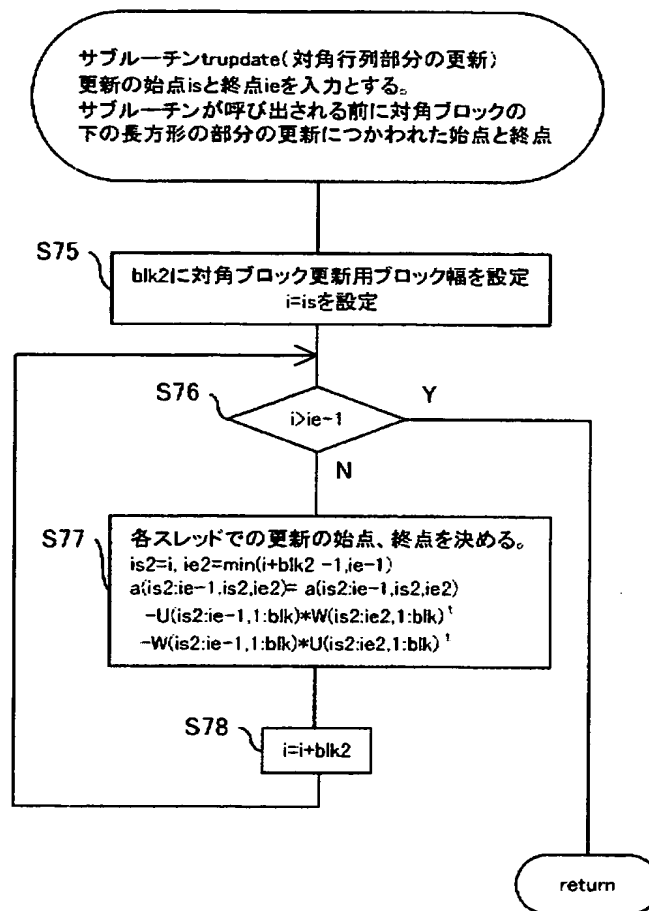
【図 23】

疑似コードの処理をフローチャートで示した図(その5)



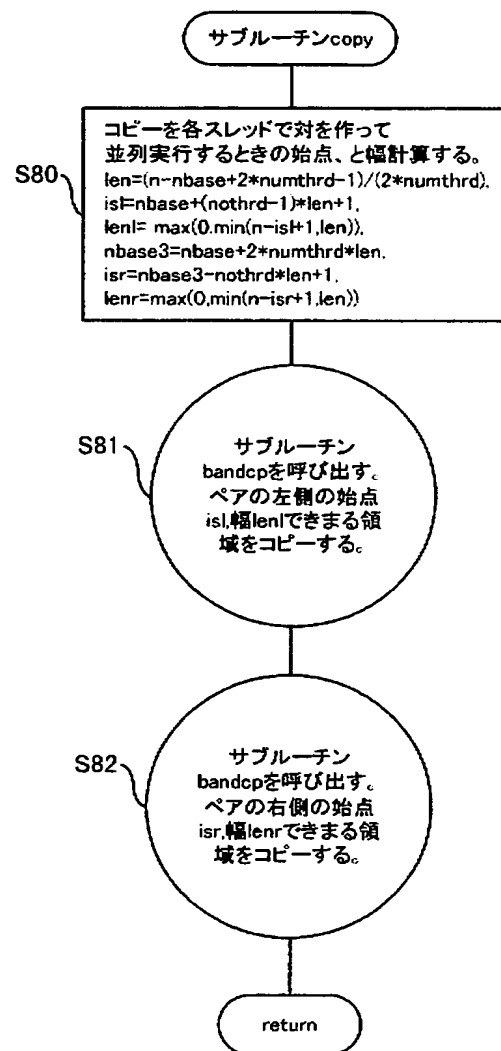
【図 24】

疑似コードの処理をフローチャートで示した図(その6)



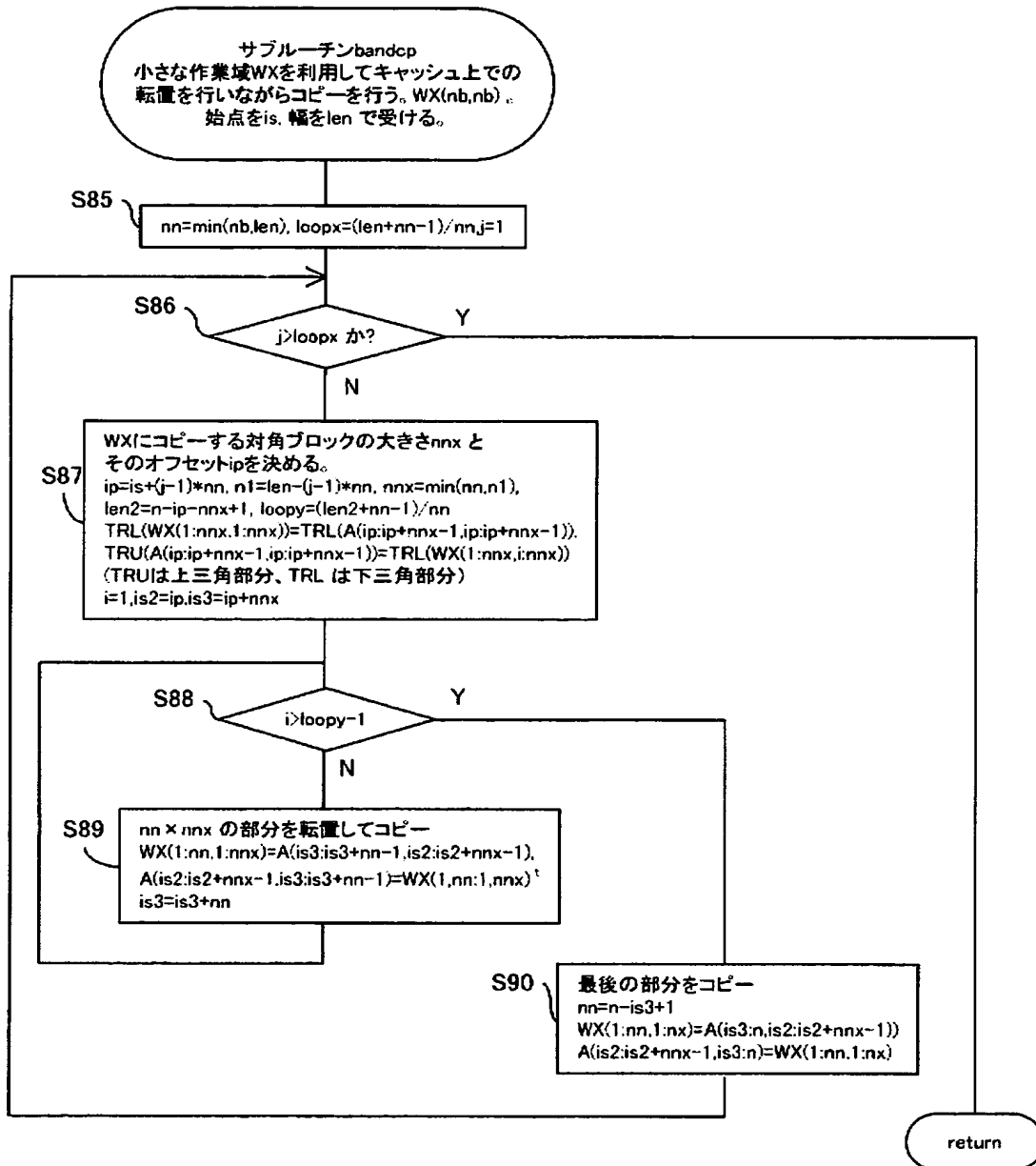
【図 25】

疑似コードの処理をフローチャートで示した図(その7)



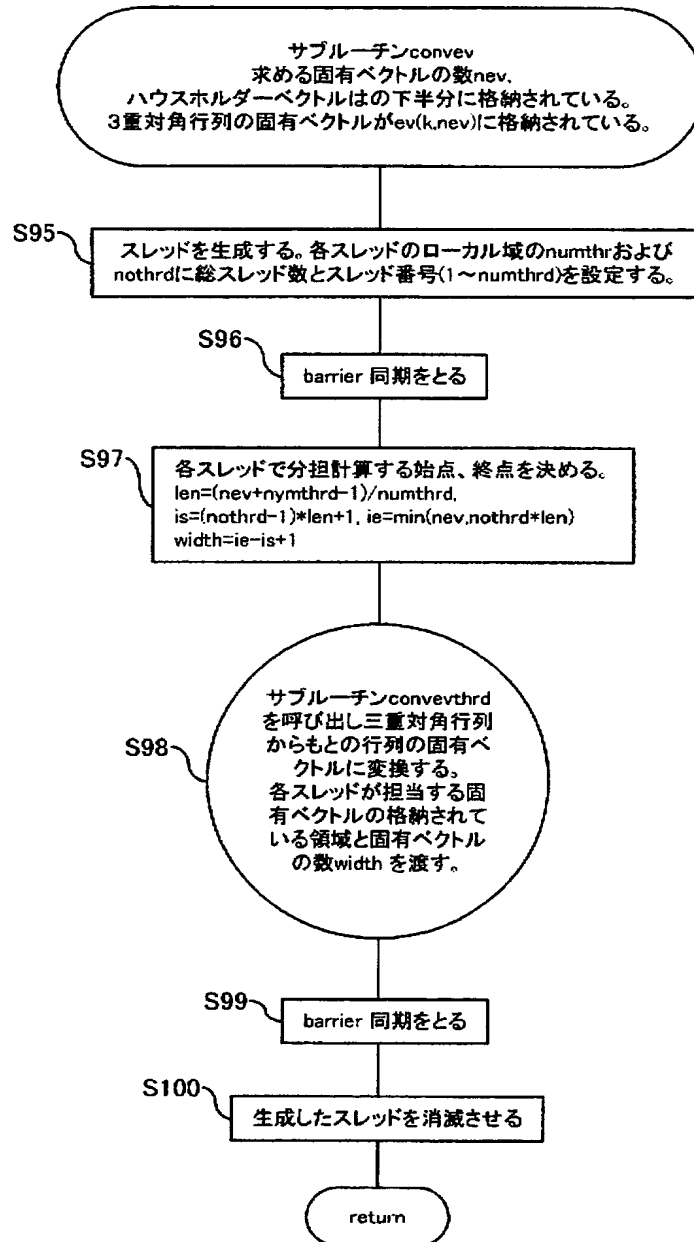
【図 26】

疑似コードの処理をフローチャートで示した図(その8)



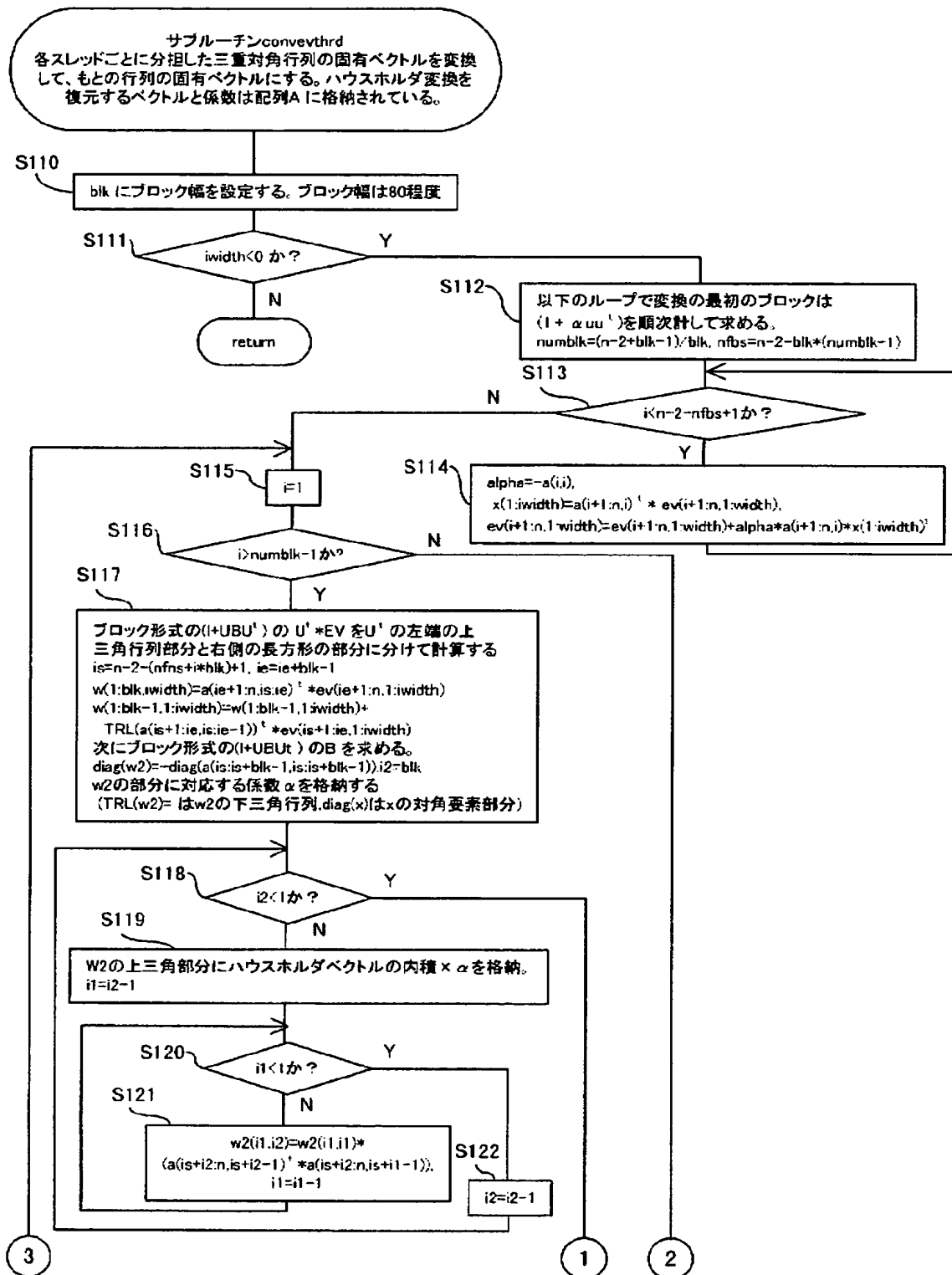
【図 27】

疑似コードの処理をフローチャートで示した図(その9)



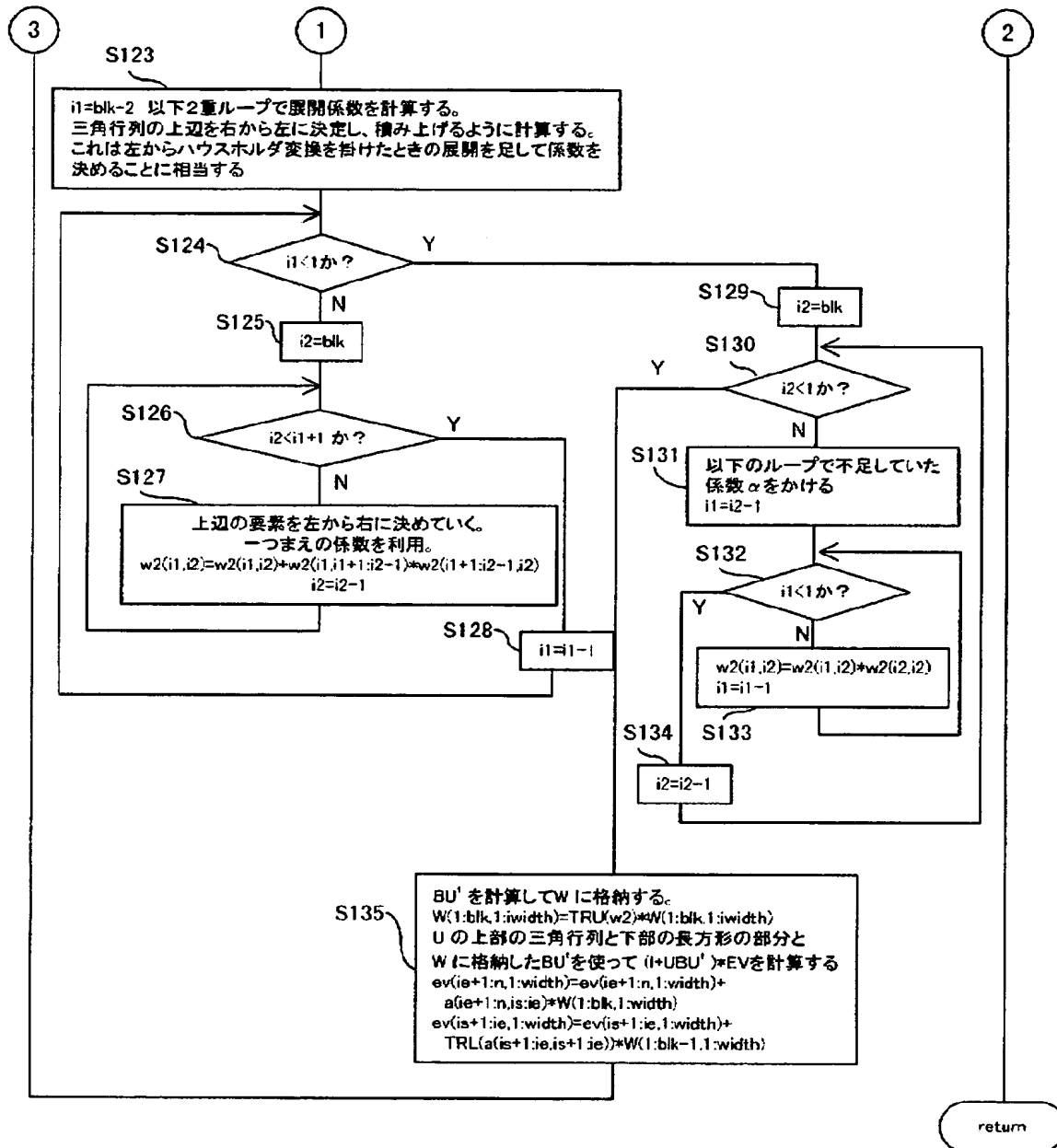
【図 28】

疑似コードの処理をフローチャートで示した図(その10)



【図 29】

疑似コードの処理をフローチャートで示した図(その11)



【書類名】 要約書

【要約】

【課題】 共有メモリ型スカラ計算機において、高速に固有値問題の演算が行える演算の並列処理方法を提供する。

【解決手段】 固有値問題を解く方法は、3段階に分かれ、行列を3重対角化するステップと、3重対角行列から固有値と固有ベクトルを求めるステップと、3重対角行列から求めた固有ベクトルを変換して、元の行列の固有ベクトルを求めるステップとからなる。特に、3重対角化を行うステップと、元の行列の固有ベクトルを求めるステップの演算負荷が大きいので、これらのステップを並列化して処理を行い、高速に固有値問題を解く。

【選択図】 図2

特願 2 0 0 3 - 0 9 2 6 1 1

出 願 人 履 歷 情 報

識別番号

[0 0 0 0 0 5 2 2 3]

1. 変更年月日
[変更理由]

1 9 9 0 年 8 月 2 4 日
新規登録

住 所
氏 名

神奈川県川崎市中原区上小田中 1 0 1 5 番地
富士通株式会社

2. 変更年月日
[変更理由]

1 9 9 6 年 3 月 2 6 日
住所変更

住 所
氏 名

神奈川県川崎市中原区上小田中 4 丁目 1 番 1 号
富士通株式会社